75445737A

CONTROL DATA

Student Training Manual for

CYBER 175 FUNCTIONAL UNITS

VOLUME 2

An Individualized Course

REVISION RECORD

REVISION	DESCRIPTION
01	Preliminary Printing
1/11/82	
A	Manual Released
5/20/82	
	Printing, 5/20/82
	Printing, 11/11/83

Publication No.
75445737

QA EDUCATION, NTHESE
1009 Washington Ave. No.
Mpls., MN 55401

© 1982, 1983
by Control Data Corporation
Printed in the United States of America

CONTENTS (Contd)

	Title	Page
Module 9: Divide Unit.		9-1
Pretest.		9-1
Learning Activities		
9-A	Programmed Text: Divide Instruction	9-3
9-B	Text Reading: Divide Primary Block Diagram.	9-11
9-C	Text Reading: Divide Detailed Block Diagram	9-16
9-D	Programmed Text: Divide Troubleshooting	9-27

FIGURES

2-1	First Adder Format.	2-7
6-1	Floating Point Format	6-4
6-2	CYBER Operand Relationship Packed and Unpacked.	6-12
8-1	Single and Double Precision Result.	8-4
8-2	Example of Large Adder Register	8-4
8-3	Floating Add Sequence	8-5
8-4	Floating Add Sequence Overflow.	8-5
8-5	Floating Add Sequence EAB	8-6
8-6	Overflow and End Around Rules	8-7
8-7	Adder Format.	8-42
8-8	Xj Plus Xk (30, 32, 34) Instructions.	8-53
8-9	Xj Minus Xk (31, 33, 35) Instructions	8-53
8-10	Signs Are Reference and Shifted Signs	8-53
9-1	Final Subtraction	9-23

TABLES

2-1	Bit, Section, and Group Generation.	2-9
6-1	Exponent Sign Bits.	6-5
6-2	Floating Point Numbers.	6-7
6-3	Examples of the Four Possible Conditions.	6-10
7-1	Shift Per/Shift Count Bits.	7-24

CONTENTS

	Title	Page
Module 10: Multiply Unit		10-1
Pretest		10-1
Learning Activities		
10-A	Programmed Text: Multiply Instructions	10-3
10-B	Text Reading: Multiply Unit Primary Block Diagrams	10-17
10-C	Text Reading: Multiply Unit Detailed Pak Diagrams	10-22
10-D	Exercise: Multiply Unit Troubleshooting	10-45
Module 11: Functional Unit Diagnostics		11-1
Pretest		11-1
Learning Activities		
11-A	Text Reading: Introduction to Diagnostics	11-3
11-B	Reference Reading: CT7 Description	11-5
11-C	Exercise: CT7 Usage	11-11
11-D	Reference Reading: CU4 Description	11-15
11-E	Exercise: CU4 Usage	11-20
11-F	Reference Reading: CT1 Description	11-23
11-G	Exercise: CT1 Usage	11-32
11-H	Reference Reading: CT1 Description	11-35
11-I	Reference Reading: MA5 Usage	11-45
11-K	Exercise: MA5 Usage	11-50
Module 12: Troubleshooting the Functional Units		12-1
Directions		12-1
Safety Practices		12-1
Learning Activities		
12-A	Lab: Troubleshooting the CYBER Model 750/760 Functional Units	12-5

FIGURES

10-1	Multiplication Matrix	10-17
10-2	Multiplication Matrix for Two Passes	10-20
10-3	Xj Multiplied by Kx (40, 41, 42 Instructions)	10-25
10-4	Pseudo Sums and Pseudo Carries	10-26
10-5	Sample Logic Modules in First Level Add	10-31
10-6	Multiply Unit 24 by 48 Matrix (Full Grid)	10-35
10-7	Flow Chart of Coefficient Multiplication	10-37
10-8	Multiply Coefficient Troubleshooting Chart	10-39
11-1	CT7 Running Display	11-7
11-2	CT7 Error Display	11-8

FIGURES (Contd)

	Title	Page
11-3	CU4 Running Display	11-17
11-4	CU4 Error Display	11-18
11-5	CT1-Running Display	11-28
11-6	CT1-Error Display	11-29
11-7	DAT-Running Display	11-37
11-8	DAT-First Error Stop Display	11-38
11-9	DAT-Second Error Stop Display (Result of BC [CR])	11-39
11-10	DAT Second Error Stop Display (Result of CN50 [CR])	11-40
11-11	MA5 Running Display	11-47
11-12	MA5 Error Display	11-48

MODULE 9
DIVIDE UNIT

Module 9 learning activities acquaint you with the logical operation of the Divide Functional Unit and review the function of the instruction.

PRETEST

Before you begin this module, sign on to the PLATO terminal and take the module 9 test. Do not waste time on answers you do not know. By taking this test, you may be able to skip some of the learning activities in this module.

LEARNING ACTIVITIES

In the Assigned column below, put a check mark by each activity assigned to you by PLM and proceed through your assigned activities. Check off each activity as you complete it. You may choose to do all of these activities or do some activities more than once.

Assigned	Completed	Activity	Description	Page
_____	_____	9-A	Programmed Text: Divide Instruction. This activity reviews the function of the Divide instruction.	9-3
_____	_____	9-B	Text Reading: Divide Primary Block Diagram. This activity traces the data bits and control flow through the 1.0 diagrams.	9-11
_____	_____	9-C	Text Reading: Divide Detailed Pak Diagrams. This activity traces the data bits and control flow through the 3.0 diagrams.	9-16
_____	_____	9-D	Programmed Text: Divide Troubleshooting. This allows you to develop some basic methods of troubleshooting the Divide Functional unit.	9-27

LEARNING ACTIVITY 9-A. PROGRAMMED TEXT: DIVIDE INSTRUCTION

This activity reviews the function of the Divide instruction. You also examine and work at least one example.

OBJECTIVE

- You will be able to translate and interpret the CPU 44 Floating Divide and 45 Round Floating Divide instructions.

FLOATING DIVIDE

The operands can be read from any two X Registers and deliver the result to a third X Register. The result is formed by dividing the contents of Xj by Xk and delivering the results to the Xi Register. If both operands are normalized, the quotient/result is also normalized. If the operands are not normalized, the quotient will not be normalized. If the operands are not normalized it is possible to have one of the following two results.

First, the quotient is correct even if there are leading zeros in the coefficient.

Second, the quotient is incorrect because the coefficient of Xj is larger than the coefficient of Xk by a factor of two or more.

Read the instruction description in section 4 and Special Case operands in section 5 of the CYBER 170 (Model 720, 730, 750, and 760) Model 176 Hardware Reference Manual, publication number 60456100.

The following are the basic rules for dividing a floating point number.

1. Only work with positive operands. If the operands are negative, complement all 60 bits. Save the original sign bits.

2. The result will be negative if the operands have unlike signs.

3. $\frac{\text{Dividend } Xj}{\text{Divisor } Xk} = Xi \text{ Quotient}$

4. The exponent of the result is the difference between the dividend and division exponent minus 60_8 or 57_8.

5. A 1-bit overflow is compensated for by adding one to the exponent and right shifting the quotient one place. The result is a normalized quantity when both the dividend and the divisor are normalized.

6. If the divisor is not normalized and the dividend coefficient is larger than the divisor by two or more, the quotient coefficient will be incorrect. The quotient is disregarded and a positive indefinite with a zero coefficient is sent to Xi. (1777 0-0).

The following is an example of Floating Divide.

o Xj = 1722 6000 0000 0000 0000
  Xk = 1722 4000 0000 0000 0000

```
Xj =  1777        Xk =  1777
     -1722             -1722
     -0055             -0055
```

```
                                          1^4000 0000 0000 0000
4000 0000 0000 0000   6000 0000 0000 0000^0000 0000 0000 0000
                      4000 0000 0000 0000
                      ---------------------
                      2000 0000 0000 0000 0
                      2000 0000 0000 0000 0
                      ---------------------
                      0000 0000 0000 0000 0
```

```
 Xj =                -55      Exponent = 1777
-Xk = (-)            -55                 -0057
                      00                  1720
 CF =                -60
                     -60
 Coef. overflow      +01
                     -57
```

Xi = 1720 6000 0000 0000 0000

The example shows that the radix point is between bit 2^0 and bit 2^{-1}. This is like double-precision in adding. This means subtracting 60_8 from the result of the difference in the two exponents, which is called the CF or Correction Factor. In this example the most significant bit, Coefficient Overflow, was lost so right shift one place and add one back to the exponent.

EXERCISE: FLOATING DIVIDE OPERATION

This exercise tests your understanding of the Divide operation.

Directions: Work the following problem. Check your answer with that given on the next page.

1. Initial conditions:

   (X1) = 6055 3677 7777 7777 7777
   (X2) = 6055 1777 7777 7777 7777

   Location 10000 - 44312 46000 46000 46000

   What are the contents of the following register after execution of the above instruction?

6055 3677

1722 4100 0000 0000 0000

1722 6000 0000 0000 0006

55
33
60

A5 4000

1777
60

1717 X3 = 5400 0000 0000 0000

ANSWER FOR FLOATING DIVIDE EXERCISE

1. (X1) = 6055 3677 7777 7777 7777
   (X2) = 6055 1777 7777 7777 7777

   Since both coefficients are negative they must be complemented.

   X1 = 1722 4100 0000 0000 0000
   X2 = 1722 6000 0000 0000 0000

   X1 = 1777 − 1722 = 0055

   X2 = 1777 − 1722 = 0055

   $\frac{X1}{X2} = X3$

```
                                                  0 ^ 5400 0000 0000 0000
6000 0000 0000 0000 | 4100 0000 0000 0000 ^ 0000 0000 0000 0000
                      3600 0000 0000 0000   0
                      ---------------------------
                      3000 0000 0000 0000   00
                      3000 0000 0000 0000   00
                      ---------------------------
                      0000 0000 0000 0000   00
```

```
 X1  =     -55          Exponent =  1777
-X2  =     -55                     -0060
            00                      1717
-CF  =     -60
           -60
```

   X3 = 1717 5400 0000 0000 0000

   This result is positive because you divided like signs.

The 45 Round Floating Divide instruction is the same as the Divide, except for the following.

The round operation consists of modifying the Xj operand by inserting a 1-bit at bit 2^{-1} as shown in the following example.

The following is an example of Round Floating Divide.

- Xj = 2000 0000 0000 0000 0004
  Xk = 2060 6000 0000 0000 0000

  Xj = 2000 Xk = 2060
  -2000 -2000
  0000 +0060

  0 ∧ 0000 0000 0000 0005
  6000 0000 0000 0000 0000 0000 0000 0004 ∧ 4000 0000 0000 0000
  3 6000 0000 0000 0000

  Xj = 0000 Exponent = 1777
  -Xk = + 0060 - 140
  - 0060 1637
  -CF = 0060
  - 0140

  Xi = 1637 0000 0000 0000 0005

CYBER 170 Model 750/760 Functional Units
Learning Activity 9-A

EXERCISE: ROUND FLOATING DIVIDE

This exercise tests your understanding of the Round Divide operation.

Directions: Work the following problem. Check your answer with that given at the end of the exercise.

1. Initial conditions:

   (X1) = 2000 0000 0000 0000 0005
   (X2) = 1722 7000 0000 0000 0000

   Location 10000 - 45312 46000 46000 46000

   What are the contents of the following register after execution of the above instruction?

X3 =

The answer to the previous exercise is on the following page.

ANSWER FOR ROUND FLOATING DIVIDE EXERCISE

1. (X1) = 2000 0000 0000 0000 0005
   (X2) = 1722 7000 0000 0000 0000

```
X1  = 2000      X2 = 1777      X1
     -2000           -1722     --=X3
      0000           -0055     X2
```

```
                                                  0 ^ 0000 0000 0000 0006
7000 0000 0000 0000  0000 0000 0000 0005 ^ 40----------------0
                                          5  2000 0000 0000 0000
```

```
 X1 =      0000          Exponent =  1777
-X2 =    - 0055                    -    3
         + 0055                      1774
-CF =    -   60
         - 0003
```

X3 = 1774 0000 0000 0000 0006

The result is positive because you divided like signs.

You have completed learning activity 9-A. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 9-B. TEXT READING: DIVIDE PRIMARY BLOCK DIAGRAM

This activity follows the data bits and control flow used by the 44 and 45 instructions through the Divide Functional Unit primary block 1.0 diagram.

OBJECTIVE

- You will be able to follow the data bits and control flow through the Divide Unit primary block 1.0 diagrams.

Directions: Place microfiche GF60420300W number 59 (8 of 10) in your microfiche viewer and select coordinate B3/B4.

The Divide instruction requires 20 clock periods to complete execution. The 44 and 45 instructions are identical except the 45 instruction adds a value of one to bit 2^{-1}. The Divide Unit operates only on positive operands. Therefore, it complements negative operands before operating on them.

INPUT REGISTERS

In quadrant A are the Xj and Xk Exponent Registers. These registers receive bits 48 through 59 from the CPU. There is also an Xj or Dividend Register, which receives bits 0 through 47 from the CPU. In quadrant C is the Xk or Divisor Register that receives bits 0 through 47. Notice that either Xj or Xk Registers complement the coefficient if the sign is negative.

CONTROL

These diagrams show only one control. Divide Busy is in quadrant C in a Divide Sequence Control. It sequences the steps in the instruction execution. This sequence control provides a static condition to distinguish each of the 19 clock periods following the issue of the Divide instruction.

METHOD

In quadrants A and C are the four Input Registers. They receive their data from the CPU operating registers during each clock period when the Divide Busy Flag is cleared. This data is used for instruction execution when a Divide instruction is issued from the CIW Register and sets the Divide Busy Flag. The Divide Busy Flag prevents the CIW

from issuing another Divide instruction for 17 clock periods following instruction issue. In the 18th clock period a second Divide instruction may be issued. Since the unit operates only on positive coefficients, each Xj and Xk Input Register will complement its data if its sign is negative.

In the upper portion of quadrant A are both Xj and Xk sign bits. This adder does a logical difference on the two bits that become your result sign. This result is sent to X Register Input Control in the CPU. The two exponents in the Exponent Register are sent to an Xj minus Xk minus CF Adder. This unit subtracts the two exponents and the correction factor if no Coefficient Overflow CF equals 60_8 and if Coefficient Overflow CF equals 57_8. This result becomes the quotient's exponent for the Xi Register and also goes to Special Case Test.

In the Special Case Test block there are three different inputs which represent the three different special case categories.

The first category consists of Overflow, Underflow, or Indefinite operand values (special case exponents) that are received from the CPU.

The second category consists of Overflow, Underflow, or Indefinite values that are formed during the exponent calculation.

The third category is Divide Fault. This applies when the dividend coefficient is greater than two times the divisor's coefficient. This will never happen if your operands are normalized.

The Divide Busy flag initiates Divide Busy in quadrant C. This starts Divide Sequence Control. It provides controls for each of the 19 clock periods following the issue of the Divide instruction. These controls then control the data movement within the Divide Unit. The remainder of the diagram is devoted to the formation of the quotient coefficient. This quotient coefficient is formed at 3 bits per clock period. The iteration network functions exactly like pencil and paper octal divide. A multiplication network forms the seven possible multiples of the divisor, FORM Xk through 7Xk, in quadrant C. In quadrant A is a Remainder Register. The Dividend or Xj is entered into this register. It also has 51 bit positions. Initially, bits 48, 49, and 50 are equal to zero. In quadrant C is the Trial Subtraction Network. This simultaneously compares each of the seven divisor multiples with the contents of the Remainder Register. The largest multiple value, whose value is less than the remainder, is subtracted from the remainder. This result is left shifted 3 binary bits or one octal position and is sent back to the Remainder Register where it is used again in the next iteration. The number of the multiple (1 through 7) picked (pick number) becomes the first quotient digit. This is in quadrant D with the PICK OCTAL DIGIT and Quotient Shift Register.

Let's do a sample divide on pencil and paper and then the same divide as the machine does it and compare each step. You will deal only with the coefficient in this example. Use the following values.

Dividend (Xj) = 4400 0000 0000 0015
Divisor (Xk) = 6000 0000 0000 0000

PENCIL AND PAPER METHOD

```
                                          0 ^ 6000 0000 0000 0021
6000 0000 0000 0000   |4400 0000 0000 0015 ^ 0000 0000 0000 0000
                      0000 0000 0000 0000
                      4400 0000 0000 0015   0
                      4400 0000 0000 0000   0
                      0000 0000 0000 0015   0
```

Skipping the next 13 as the remainder is smaller than the divisor.

```
                                15   0000 0000 0000 000
                                14   0000 0000 0000 000
                                 1   0000 0000 0000 0000
                                     6000 0000 0000 0000
                                     2000 0000 0000 0000
```

Let's now look at how the machine does it. The following would be the contents of the multiplication network.

```
 Xk =    6000 0000 0000 0000
2Xk = 1 4000 0000 0000 0000
3Xk = 2 2000 0000 0000 0000
4Xk = 3 0000 0000 0000 0000
5Xk = 3 6000 0000 0000 0000
6Xk = 4 4000 0000 0000 0000
7Xk = 5 2000 0000 0000 0000
```

The dividend (Xj) enters the Remainder Register as follows:

REMAINDER = 0 4400 0000 0000 0015

Initially, the Quotient Shift Register contains XXXX XXXX XXXX XXXX.

The next 17 iterative operations are performed as follows:

1. Compare REMAINDER vs MULTIPLICATION NETWORK Xk through 7Xk. REMAINDER is smaller than any of them so PICK OCTAL DIGIT equals 0. Enter this 0 in the Quotient Shift Register. It is left shifted one octal digit each clock period.

   QUOTIENT SHIFT REGISTER = XXXX XXXX XXXX XXX0
   REMAINDER = 4400 0000 0000 0015 and left shift three or one octal position.
   REMAINDER REGISTER = 4 4000 0000 0000 0150
   Enter 0 when ↑ left shift three.

2. Compare REMAINDER vs MULTIPLICATION NETWORK Xk through 7Xk. 6Xk is the largest number that is smaller than the REMAINDER. PICK OCTAL DIGIT equals 6. Enter this 6 in the Quotient Shift Register as it left shifts one octal digit.

   QUOTIENT SHIFT REGISTER = XXXX XXXX XXXX XX06
   REMAINDER = 0000 0000 0000 0150 and left shift three.
   REMAINDER REGISTER = 4) 0000 0000 0000 1500

3. For the next 12 iterations, the REMAINDER is always smaller than the MULTIPLICATION NETWORK values, so PICK OCTAL DIGIT equals 0. The Quotient Shift and Remainder Registers have both been left shifted 12 more octal places and zeros have been entered.

   QUOTIENT SHIFT REGISTER = XX06 0000 0000 0000

14. REMAINDER REGISTER = 0 1500 0000 0000 0000

15. Compare REMAINDER vs MULTIPLICATION NETWORK Xk through 7Xk. REMAINDER is smaller than any of them so PICK OCTAL DIGIT equals 0. Enter this 0 in the Quotient Shift Register. It is left shifted one octal digit.

    QUOTIENT SHIFT REGISTER = X060 0000 0000 0000
    REMAINDER = 1500 0000 0000 0000 and left shift three
    REMAINDER REGISTER = 1 5000 0000 0000 0000

16. Compare REMAINDER vs MULTIPLICATION NETWORK Xk through 7Xk. 2Xk is the largest number that is smaller than the REMAINDER. PICK OCTAL DIGIT equals 2. Enter this 2 in the Quotient Shift Register as it left shifts one octal digit.

    QUOTIENT SHIFT REGISTER = 0600 0000 0000 0002
    REMAINDER = 1000 0000 0000 0000 and left shift three
    REMAINDER REGISTER = 1 0000 0000 0000 0000

17. Compare REMAINDER vs MULTIPLICATION NETWORK Xk through 7Xk. 1Xk is the largest number that is smaller than the REMAINDER. PICK OCTAL DIGIT equals 1. Enter this 1 in the Quotient Shift Register as it left shifts one octal digit.

    QUOTIENT SHIFT REGISTER = 6000 0000 0000 0021
    Discard the remainder (2000 0000 0000 0000) and transmit the Shifted Quotient Register to the Result or Xi Register.

You have completed learning activity 9-B. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 9-C. TEXT READING: DIVIDE DETAILED PAK DIAGRAMS

This activity follows the data bits and control flow through the Divide Functional Unit detailed 3.0 diagrams.

OBJECTIVE

- You will be able to follow the data bits and control signals through the Divide Unit detailed 3.0 diagrams.

Directions: Place microfiche GF60420300W number 59 (8 of 10) in your microfiche viewer and select coordinate D1/D2.

Since the diagrams used in Divide have most of one item on each diagram, we will cover each as completely as possible.

The 3.0 diagrams are mainly Exponent and Special Case.

INPUT REGISTERS

In quadrant A is a KR module at 7P02. It has the Input Registers for both operands' exponents, bits 48 through 59. These are clear/enter type registers. The data that is in the registers is held for 17 clock periods while the Divide Busy Flag is set. Bit 59 of both Xj and Xk are checked and if either are negative it will complement its respective operand. Both sign bits go to quadrant B to a Logical Difference Adder on a DP module at 7P07. Remember logical difference is

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ \overline{0} & \overline{1} & \overline{1} & \overline{0} \\ 0 & 0 & 1 & 1 \end{array}$$

and in division like signs give you a positive result and unlike signs give you a negative result. This result is sent to X Sign Control in the CPU. In quadrant A and the module 7P02, the full exponent with a positive value goes to a DO module at 7P08. Using the example of a Divide on page 9-4, the exponents were Xj = 1722 and Xk = 1722. There was also Coefficient Overflow in this example.

Take the dividend first or Xj. The bias is removed making it equal to 0055 then complemented to 7722. Since bit 59 is not used at this time, Xj equals 3722. Xk equals 1722 and, with its bias removed, Xk equals 0055. The other input is Overflow that equals one so it will gate in 57_8 to the Preliminary Adder. Overflow comes from the DS module at 7P03 in quadrant D. With Overflow, the Pick Number, the

bottom input, would equal one. This would be complemented to a zero. The bottom output would be complemented again to a one and go to quadrant C, DQ module at 7P06. This would happen at CP03 or clock three. It leaves Divide Sequence Control at clock four as a one and gates in 57_8.

Value 60_8 is blocked because overflow was complemented to zero. These bit Enables and bit Carries go to the Final Adder on the DP module at 7P07 in quadrant B. The result would be the following values.

```
    Xj = 1777
         1722
Unbiased 0055 complemented to 7722 no bit 59 so 3722

    Xk = 1777
         1722
Unbiased 0055                          no bit 59    0055

                                                    3777

Correction Factor with Coefficient Overflow          57
                                                   ------
                                                    0056
                                              EAB      1
                                                   ------
                                                    0057
```

Bias is added so bit 2^{10} is set making $\overline{\text{Exponent}}$ equal to 2057. This goes to the DS module at 7P03 and to an AND function, to the left of the Output Register. The Gate Quotient is the second input arriving at CP18 or clock 18. The exponent would be held in the Output Register until CP19 or clock 19. The exponent would then be gated through the AND NOT function giving you a true value. So 2057 is complemented to 1720. Bit 59 is zero. It is added now. This diagram does not show it as the pak diagram does, but with Gate Quotient equal to one, CP19 making up bit 59 is equal to zero. So the exponent value going to the CPU is equal to 1720. This agrees with the example.

SPECIAL CASE

Learning activity 9-B explained three special case categories. The first category was special case exponents received from the CPU. The second category was special case exponents formed within the unit during exponent calculations.

The third category was Divide Fault when the dividend (Xj) was two times greater than the divisor (Xk).

The first category is taken care of in quadrant A on the DO module at 7P08. If Xj or Xk Overflow, Underflow, or Indefinite, it will cause a one to the DP module at 7P07 in quadrant B. This module also takes care of the second category. If Overflow, Underflow, or Indefinite is generated in the final add it will be detected here.

Category three was Divide Fault. If it exists, the quotient would be two or bit 1 equal to a one. In quadrant D the DS module at 7P03, the bottom input (the Pick Number) has bit 1 equal to a one. It is complemented to a zero and complemented again to one to form Divide Fault bit 1. This goes to quadrant C to the DQ module at 7P06. Divide fault enters at clock three or CP03 and leaves Divide Sequence Control at clock four or CP04. This goes to quadrant B and the DP module Special Case Check with its two outputs. The top one sends Overflow, Underflow, and Indefinite Condition Flags to the CPU. The bottom command is Special Case that goes to the DQ module at 7P06 in quadrant C at clock 18 or CP18. GO DIV SPL Case equals a one and read note (1). GO DIV SPL Case is complemented becoming a zero. This blocks all outputs, causing all ones to be sent to the CPU. The other part of GO DIV SPL Case equals a one and goes to the CPU. The result in the Xi Register is determined by the Error Flag. These results can be found in section 5 of the hardware reference manual, table 5-5. Special condition flags also cause appropriate error condition flags to be set in the PSD Register.

COEFFICIENT

Directions: Select coordinate E1/E2, the Divisor Multiplication Network.

3.1

Learning activity 9-B showed logic that develops the seven possible multiples of the divisor, called FORM Xk through 7Xk. The divisor coefficient Xk bits 0 through 47 enter in quadrant A to four DJ modules at 7P09, 7P11, 7P13, and 7P15. When ENTER Xk is equal to a one, new data is received every clock period until clock one or CP01 of a Divide instruction. It remains a zero until clock 18 or CP18 and new data may enter. The sign bit, bit 59 is used to complement the data if it is negative. The rules only work on positive operands. The remaining modules form the seven multiples.

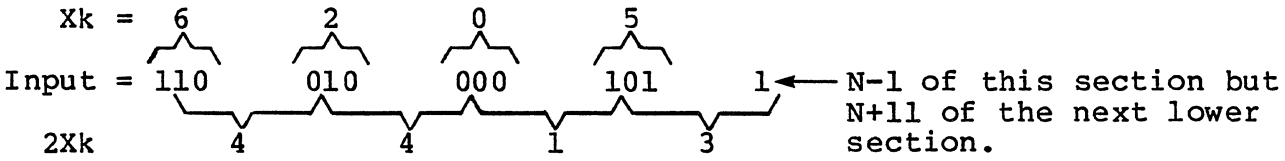
In this explanation let's use the multiplication network contents used in the pencil and paper example used in learning activity 9-B on page 9-13.

```
 Xk =   6000 0000 0000 0000
2Xk = 1 4000 0000 0000 0000
3Xk = 2 2000 0000 0000 0000
4Xk = 3 0000 0000 0000 0000
5Xk = 3 6000 0000 0000 0000
6Xk = 4 4000 0000 0000 0000
7Xk = 5 2000 0000 0000 0000
```

This logic is made up of two adders 3Xk and 5Xk, one subtractor 7Xk and four networks Xk, 2Xk, 4Xk, and 6Xk. Remember, the network always develops 50 bits, 0 through 49.

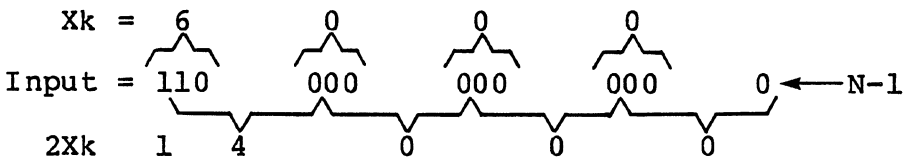
The DK modules at 7K04, 7K07, 7K10 and 7K13 in quadrant A and B are called the 3Xk Adder logic. Xk bits 0 through 47 enter these four modules. Using 7K07 as the example, they take the top path and become Xk multiple bits 12 through 23. Each module shows only 12 bits (a section) of Xk but shows 13 bits going to the Preliminary Adder as on 7K07. Actually there are 13 incoming bits which are called N-1 through N+11 for each module. The 2Xk is formed by a left shift of one bit position. (See NOTE on diagram.) This agrees with our pencil and paper example. 6000 left shifted one would equal 1 4000.

Let's look at this a little closer. The 13 bits received are bits N-1 through N+11. N-1 through N+10 will be used as 2Xk bits N through N+11. In other words, N+11 in one section is N-1 in the next higher section. An example of this is shown below.



Used as N-1 of the next higher section but N+11 of this section.

In our example of 6000, it would look like this.



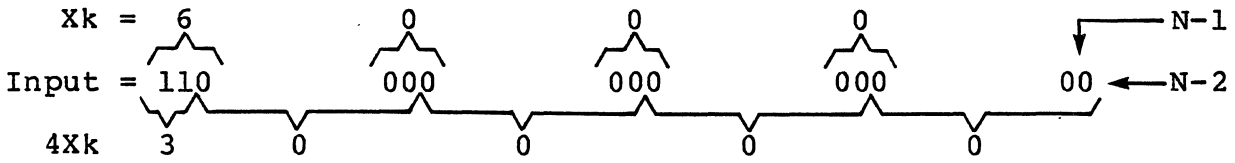
So it was actually a hardwired left shift.

3Xk is formed on the four DKs and the one DN module by adding Xk and 2Xk. From the example this would be as shown below.

```
 Xk  =   6000 X----------X
2Xk  =  14000 X----------X
3Xk  =  22000 X----------X
```

The Preliminary Adder generates Section Carries and Section Enables. These go to the DN module at 7K16. These Carries and Enables are added in the section adder. The result are four Section Carries In, three of which go back to the DK modules at 7K07, 7K10, and 7K13. These Section Carries are added to the result of the Preliminary Add resulting in 2Xk plus Xk equals 3Xk bits 0 through 47. At the DN module at 7K16 in quadrant B and the top section add, take the last Section Carry, Section 4 Carry. This goes to a final add that has Xk bit 47 coming in. The result of this add is 3Xk bits 48 and 49 that are used in the iteration network to form the result.

The next logic covered is the 5Xk adder. It is on four DL modules 7K05, 7K08, 7K11, and 7K14 in quadrant A and the DN modules at 7K16 in quadrant D. 5Xk is formed by adding 4Xk plus Xk. 4Xk is formed by a hardwire left shift two of Xk. It would look like the example below.



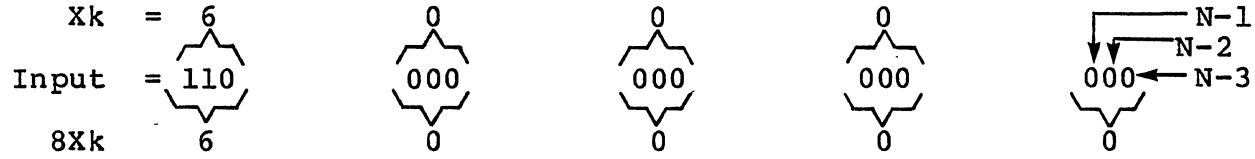
Used on the next higher section/module as N-1 and N-2

5Xk was 4Xk plus Xk which would be as follows:

```
 Xk  =   6000 X----------X
4Xk  =  30000 X----------X
5Xk  =  36000 X----------X
```

Looking at the logic on the DL modules in quadrant A and using 7K08 for the example, Xk bits 10 through 23 enter. They are left shifted two places to form 4Xk and add Xk in the Preliminary Adder. The Section Carries and Section Enables go the DN module in quadrant D. These go through a section Adder with a result of four Section Carries In, three of which go to the DL modules at 7K08, 7K11, and 7K14. These are added to the result of the Preliminary Add from 4Xk plus Xk giving a result of 5Xk bits 0 through 47. At the 5Xk Adder on the DN module in quadrant D, take the last section Carry In and add with Xk bits 46 and 47. This results in 5Xk bits 48, 49, and 50.

The next logic covered is the 7Xk Subtractor, which is on four DM modules at 7K06, 7K09, 7K12, and 7K15 in quadrant C and the DN module in quadrant D. To form 7Xk, subtract Xk from 8Xk. 8Xk is formed by a hardwired left shift three of Xk. Looking at our example, it would look like the example below.



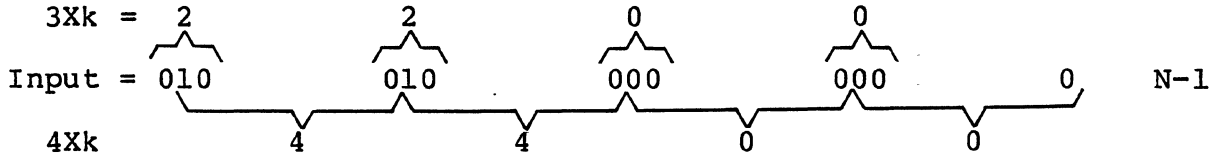
Used on the next higher section/module as N-1, N-2, and N-3.

7Xk was 8Xk minus Xk which would be as follows:

```
8Xk = 6  0000 X----------X
-Xk =    6000 X----------X
7Xk = 5  2000 X----------X
```

Looking at the logic on the DM modules in quadrant C and using 7K09 for the example, Xk bits 9 through 23 enter. They are left shifted three places to form 8Xk and subtract Xk in the Preliminary Subtractor. The Section Borrows Out and Section Enables go to the DN module in quadrant D. They go through a section subtractor with a result of four Section Borrows In, three of which go back to the DM modules at 7K09, 7K12, and 7K15. This is subtracted from the result of the Preliminary Subtraction from 8Xk minus Xk giving a result of 7Xk bits 0 through 47. At the 7X Subtractor on the DN module in quadrant D, take the last Section Borrow In and subtract Xk bits 45, 46, and 47. This results in 7Xk bits 48, 49, and 50.

The remaining multiple is 6Xk. This is 3Xk left shifted one bit position using the example below.



The logic for 6Xk is found on the 3.2 diagrams at coordinate F1/F2. In quadrant A are DA and DH modules. Using the DA module at 7M07 there is a rank 6 adder. Using the DH module at 7L15 the second input from the top is 3Xk bits 18 through 20 to its rank 3 and 6Xk bits 17 through 19 to its rank 6. This completes the forming of the Xk multiples.

Iteration or Coefficient Network is the logic that develops the result. This network forms the quotient/answer of two 48-bit integers. The dividend is Xj, the divisor is Xk, and the quotient is Xi. Use the pencil and paper method and machine method as found on pages 9-13 through 9-15 for the following explanations. The following networks form the quotient coefficient.

Input Register	Receives Divide operand coefficients.
Multiplication Network	Forms multiples of divisor Xk through 7Xk.
Trial Subtraction Network	Subtracts multiples of divisor from the dividend (first iteration) or from the remainder (all other iterations) to determine which of the multiples of Xk through 7Xk are larger than the dividend or the remainder.
Partial Subtraction Network	Subtracts multiples of divisor from dividend (first iteration) or from current remainder (all other iterations) to begin forming the next remainder.
Final Subtraction Network	Performs final subtraction of the largest multiple that is smaller than the current remainder. The output is a quotient digit (pick number) and the new remainder.

Trial Subtraction Network subtracts each of the seven multiples of the divisor (Xk) from the dividend (Xj) (first iteration) or the remainder (all other iterations). If the subtraction produces an End-Around Borrow (EAB), that multiple is larger than the remainder (or dividend). The Divide Unit senses all seven EAB bits to determine which, if any, of the multiples are larger than the remainder.

The first iteration takes place at clock two or CP02. For the operands chosen in this iteration, all EAB bits are equal to ones. This means that all seven multiples are larger than the dividend. In this case, a zero is the pick number and becomes the first quotient digit. The remainder is left shifted one octal digit (three bit positions) and goes to the Remainder Register. This is the same as adding a zero to the right of the dividend as in the pencil and paper method.

At clock three or CP03, the second iteration takes place. Each of the seven multiples are subtracted from the remainder. The seven EAB bits are sensed by the Final Subtraction Network. In this iteration, the EAB bits 0 through 6 are zeros and EAB bit 7 is equal to a one. A pick number of six would be generated and becomes the second quotient digit. This would continue for 17 iterations or CP18.

PARTIAL SUBTRACTIONS NETWORK

In the pencil and paper Divide, 6Xk was subtracted from the remainder. In the Divide Unit, the remainder is complemented and added to 6Xk. The result is complemented to produce the difference. An example would be the second iteration of the pencil and paper example.

```
Remainder complemented = 3377 7777 7777 7762 7
Added to 6Xk             4400 0000 0000 0000 0
                         7777 7777 7777 7762 7

Complement to produce
  difference             0000 0000 0000 0015 0
```

The Partial Subtraction Network produces inclusive bit Enables for rank 1 through 7. The inclusive bit Enables for rank 0 are the true remainder bits.

FINAL SUBTRACTION NETWORK

The Final Subtraction Network adds the Group Borrow Inputs from the Trial Subtractor Network to the ranks 1 through 7 inclusive bit Enables from the partial subtraction network and the rank 0 inclusive bit Enables. An example of this would be the paper and pencil computation shown in figure 9-1.

Divide Unit		Pencil and Paper
REMAINDER		REMAINDER
+ 6Xk		- 6Xk
Inclusive Bit Enables	Results of Partial Subtraction Network	
+ Group Borrows	Results of Trial Subtraction Network	
DIFFERENCE	Results of Final Subtraction Network	
DIFFERENCE	Completed to become Difference	DIFFERENCE

Figure 9-1. Final Subtraction

The logic for Trial Subtractor Network is divided into first and second stages. First stages are in a quadrant A on the DA modules at 7M01 through 7M15. They are ranks 3, 6, and 7, bits 3 through 47. The uppermost module is a DD module at 7M16 for ranks 3, 6, and 7, bits 48 through 50. In quadrant C on the DB modules at 7N01 through 7N15 are ranks 1, 2, 4, and 5, bits 3 through 47. The uppermost module in that group is a DE module at 7N16 for ranks 1, 2, 4, and 5, bits 48 through 50. In quadrant C is a DC module at 7L01 that has ranks 1 through 7, bits 0 through 2. This network has 3 bit groups or one octal digit. In the clock period following the instruction being issued (CP01) the two operands are received from the CPU. One input is the dividend on the first iteration and the remainder after that. Let's look at how the dividend gets to the Remainder Register.

In quadrant B are four DI modules at 7P10, 7P12, 7P14, and 7P16. Using 7P12 as our example, the Xj coefficient is received in the Dividend Register along with the sign bit, bit 59. If negative or equal to a one this data will be complemented. Since the Divide Unit is at clock one or CP01, Circulate Remainder equals a zero. This is complemented allowing Xj to be entered, through the OR function, and complemented as shown in figure 9-1. This is sent to the DA and DD modules in quadrant A. Bits 48 through 50 on the DD module are initially set to zeros. The DB and DE and DC modules in quadrant C receive the same data. Bits 48 through 50 on the DE module are also initially set to zero.

Using the DB module at 7N07 for our example, the bottom input bits 21 through 23 from the DI module are gated by Xj into the OR function and then to the Subtractor. To get this data through the Subtractor without changing its value, go to coordinate D1/D2 and the DQ module in quadrant C. The fourth output from the bottom (RELEASE REMAINDER) is equal to zero on CP01 or first iteration. This goes back to the DD and DE modules on the 3.2 diagrams on coordinate F1/F2. At the DD module in quadrant A, the RELEASE REMAINDER equals a zero. This is complemented to a one causing all EABs to equal a one. This results in a pick number equal to zero. This means the dividend passes through the Final Subtraction Network unaltered.

The second input is the Multiplier Xk through 7Xk. Between the group of DAs and DBs in quadrant A and C is a line labeled Xk-7Xk DIV 3.1 Multiplication Network. Follow this to the right and it branches up with 3Xk, 6Xk, and 7Xk. The next path is Xk, 2Xk, 4Xk, and 5Xk. Each goes to its respective rank 1 through rank 7 First Stage Trial Subtractor. This generates Group Enables and Group Borrows Out. This completes the first stage trial.

SUBTRACTION NETWORK

These Group Borrow Outs and Group Enables go to quadrant D and the DG modules at 7006 through 7012 and DF modules at 7L06 through 7L12. The second half subtraction is completed, resulting in an EAB for each of the 7 ranks and Group Borrows Out for groups 3 through 16.

At this time the unit has generated the rank EABs so you know which Multiplier is the largest of the seven that goes into the dividend or remainder. Also Group Borrows Out have been generated and these will be used during the final subtraction which results in a remainder for that iteration.

PARTIAL SUBTRACTION NETWORK

The dividend, if it is the first iteration or Remainder, and all seven multiples must go through a partial subtraction. This logic is located in quadrants A and B at DH modules at 7L02 through 7L05, 7L13 through 7L16, 7002 through 7005 and 7013 through 7016. Taking the remainder first from the DA modules in quadrant A, bits 3 through 47, and the DC module in quadrant C, bits 0 through 2, it enters the DH modules as shown at the bottom of the stack. This remainder is in complement form or Remainder. The seven multiples Xk through 7Xk enter at the bottom as well. This results in generating inclusive bit enables for all seven ranks/multiples.

The outputs of the partial subtraction network are left shifted one octal position (three binary bits). Ranks 1 through 7 Inclusive Bits 3 through 50 Enables leave the DH modules. Let's stop and return to the DH module at 7L15.

The bottom line is $\overline{\text{Remainder}}$ through a complementer and this is rank 0 Remainder in true form, bits 0 through 47. These bits go to the four DI modules in quadrant B. These bits go through a hardwire left shift of three bit positions to an AND function with Circulate Remainder equal to a one. These rank 1 bits go up and over and now all ranks, ranks 0 through 7, and bits 3 through 7 Enables go to the DA and DD modules in quadrant A and the DB and DE modules in quadrant C.

Using the DB module at 7N07 as an example, one input would be Group Borrow Inputs from the Second Stage Trial Subtractor modules in quadrant D. Another input, as just covered, is the Inclusive Bit Enables from partial subtraction. The rank that would be selected is controlled by the EABs bits from the Second Stage Trial Subtractor in quadrant D. The result of this subtraction would be stored in the Remainder Register. It would then start all over on the next iteration.

In the pencil and paper example and the second iteration, the divisor went into the remainder six times. In this case rank 6 would have been the final subtraction, as it was the highest rank that did not generate an EAB. This rank number of 6 would go to the DD module in quadrant A. The pick an octal digit 0 through 7 logic senses the EAB selecting the largest rank without an EAB, and generates a pick number. This goes to the Quotient Holding Register and is sent to the Coefficient Shift Register on the DS module at 7P03 in quadrant D. These are on the 3.0 diagrams at coordinates D1/D2. This module holds bits 0 through 2 and 45 through 47 of the coefficient. Below this are two DR modules at 7P04 and 7P05. They hold bits 3 through 44.

Each clock period or iteration starting at CP03 this quotient is left shifted three bit positions. Three new bits are received from the DD module on the 3.2 diagrams. At this same time the upper three bits 45 through 47 are discarded. After 17 iterations, the Shift Register holds the 48-bit quotient. At CP18, Gate Quotient goes to the Flip Flop, so at CP19 it will make all the AND NOT functions complement the result/quotient and send it to the Xi Register in the CPU. If the result had Coefficient Overflow then all the output bits would go through the right shift one logic.

This completes a normal Divide operation. If something is not clear please review that area.

ROUND DIVIDE

For this you will look only at the areas that have not been covered. Use the 3.0 diagrams at coordinate D1/D2.

The XK module in quadrant C, the bottom input is m0. It equals a one when doing a Round instruction that equals 45. At CP02 Round (bit 2) goes to the DI module at 7P10 at quadrant B on the 3.2 diagrams at coordinate F1/F2.

During the Round instruction, a round bit is added to the dividend to increase its value by one-half. The effect this has on the result/quotient varies depending on the value of the divisor and dividend.

You have completed learning activity 9-C. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 9-D. PROGRAMMED TEXT: DIVIDE TROUBLESHOOTING

This activity allows you to develop approaches to troubleshooting the Divide Unit.

OBJECTIVE

- You will be able to use the oscilloscope to find failing bits and picks.

This activity gives you a procedure in troubleshooting problems in the Divide Functional Unit.

This procedure is only one of the ways of troubleshooting the divide functional unit.

First are some of the more important locations to look at when DAT fails.

Location	10 = 44	This is the 44 instruction. Zero out the instruction you do not want to run.
	11 = 45	This is the 45 instruction
	13 = X	If equal to a one it will loop on the error
	64 =	The five possible test sections. Loads with a 37.
Location	100 =	Xj Operand
	101 =	Xk Operand
	102 =	Failing Instruction
	103 =	Expected Result
	104 =	Actual Result
	105 =	The failing bits
	106 ↓ 114 =	The multiplier of the Xk coefficient (Divisor) an example of this can be found on page 9-13 in learning activity 9-B.
	117 ↓ 137 =	Picks and remainders from each iterations. Examples of how these were formed can be found on pages 9-14 and 9-15 in learning activity 9-B.

The following is how to set up the oscilloscope:

1. Sync externally on 6K06 Test Point 62-Divide Busy.

2. Channel A on 7P06 Test Point 6, these are the divide iterations.

3. Channel B will be used to look at the pick number and the value of the remainder. These can be found at the following locations.

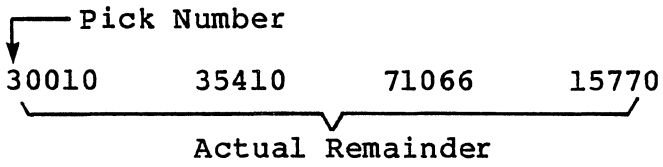
   The pick number can be seen on 7M16
   Test point 13 equals pick bit 0
   Test point 12 equals pick bit 1
   Test point 11 equals pick bit 2

   The remainder value can be seen on the DA's and DB modules
   Test point 33 equals bit 0
   Test point 31 equals bit 1
   Test point 32 equals bit 2

Remember, you are looking at negative logic so ones on the scope are zeroes and zeroes are actual ones.

Locations M01 and N01 hold the least significant bits so progressing to the left would bring you to the most significant bit.

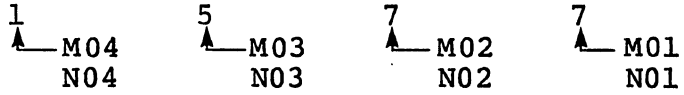
Assume the number in location 133 was equal to:



The most significant digit 3 is the pick mumber. The least significant digit 0 is discarded, so the remaining digits is the actual remainder for that iteration.

The DA and DB modules always contain the same value when the divide is working properly. So to make sure we know where the remainder values are look at these following examples.

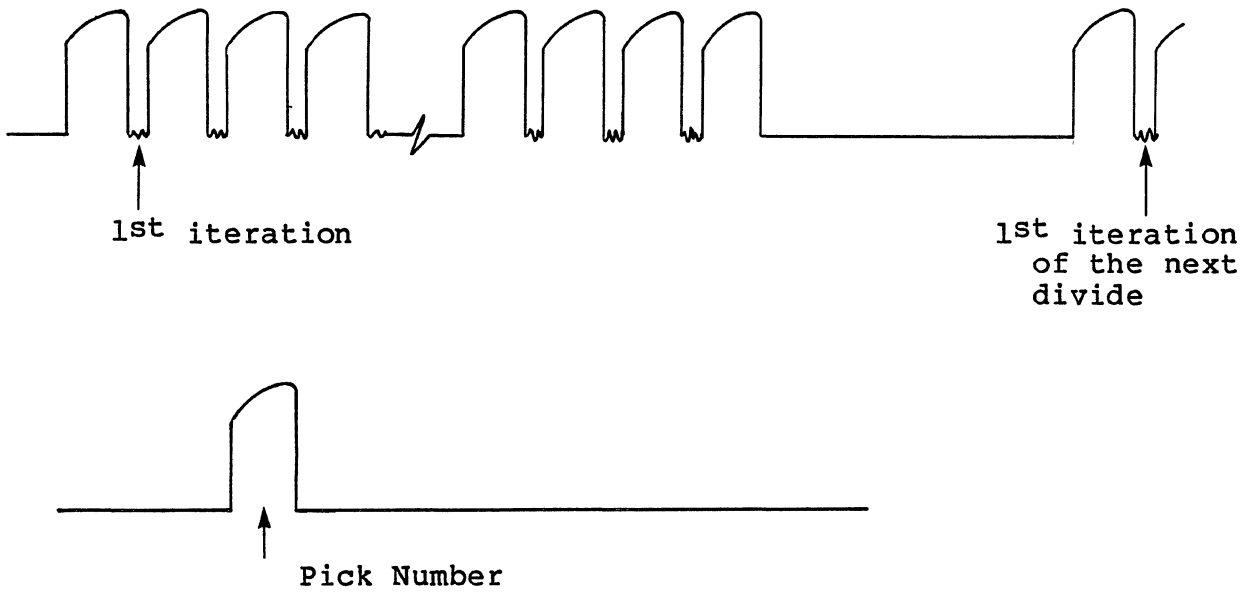
The lower four digits would be found on these modules.



When DAT fails you have all the information on the failure. Get your operands as simple as possible.

The first thing to look for is a bad pick number. Then back up an iteration looking for a good pick number. At this point look at the remainder. When you find a bad bit compare it against its companion module. If it is the same value, drop back another iteration. The failing bits should be three bits lower, as the divide is done three bits at a time. When the failing bit is found again compare it between the DA and DB. If a difference is found between the DA & its companion DB module look to see which has the right remainder value. Change the module which has wrong value.

The following is how iterations and the data will look on the scope.



Because of timing, do not use the first iteration when looking at the data, start with the second as shown above. The value falls between the second and third iteration in this example and it is equal to a one.

So to see all the data that is in location 117 of DAT you would use the second iteration as seen on scope. To see data in location 123 use the sixth iteration and so on.

The following is an actual failure of a bad DA in 7M02.

Location in DAT	Value in that location			
133	30010	35410	71066	15770
134	20050	46220	53174	73620
135	10015	57252	13115	04150
136	30020	61677	54631	23370

When scoping the pick number on the 17th iteration which is the same as what is in location 136 was a one, not three. When we dropped back to the 16th iteration, we had a pick number of one. We now start scoping the remainder that is shown in location 135. M01 showed a 5, M02 had a 1, M03 had a 4, and M04 had a 3. The DB at N04 also had a 3. When we drop back to the 15th iteration and use the remainder that is shown in location 134, M01 had a 2, M02 had a 6, and M03 had a 1. The DB at N03 also had a 1. On the 14th iteration, M01 had a 7, and M02 had a 0; N02 had a 7. We changed the DA at M02.

The preceding are not hard cold facts as they depend on the operands. If the problem is in the DAs or DBs, the answer usually can be found by looking at pick numbers and/or data. In the other modules, you have to find a bad value and, using that information, swap out module that would hold those bits.

8959H

MODULE 10
MULTIPLY UNIT

Module 10 learning activities acquaint you with the logical operation of the Multiply Functional Unit and review the function of the instruction.

PRETEST

Before you begin this module, sign on to the PLATO terminal and take the module 10 test. Do not waste time on answers you do not know. By taking this test, you may be able to skip some of the learning activities in this module.

LEARNING ACTIVITIES

In the Assigned column below, put a check mark by each activity assigned to you by PLM and proceed through your assigned activities. Check off each activity as you complete it. You may choose to do all of these activities or do some activities more than once.

Assigned	Completed	Activity	Description	Page
_____	_____	10-A	Programmed Text: Multiply Instructions. This activity reviews the function of the multiply instruction.	10-3
_____	_____	10-B	Text Reading: Multiply Unit Primary Block Diagram. This activity traces the data bits and control flow through the 1.0 diagrams.	10-17
_____	_____	10-C	Text Reading: Multiply Unit Detailed Pak Diagrams. This activity traces the data bits and control flow through the 3.0 diagrams.	10-22

Assigned	Completed	Activity	Description	Page
_____	_____	10-D	Exercise: Multiply Unit Trouble-shooting. This allows you to develop some basic methods of troubleshooting the Multiply unit.	10-44

LEARNING ACTIVITY 10-A. PROGRAMMED TEXT: MULTIPLY INSTRUCTIONS

This activity reviews the function of the Multiply instruction. You will also examine and work at least one example.

OBJECTIVE

- You will be able to translate and interpret the CPU 40 Floating Product, 41 Round Floating Product, and 42 Floating Double Precision Product.

FLOATING PRODUCT INSTRUCTION

The operands can be read from any two X Registers and deliver the results to a third X Register. The result is formed by multiplying the contents of Xj and Xk and delivering the results to the Xi Register. These operands are not necessarily normalized. If both operands are normalized, the result will be normalized. If either or both operands are not normalized, then the result may or may not be normalized.

Read the instruction description in section 4 and Special Case Operands in section 5 of the CYBER 170 (Model 720, 730, 750, and 760) Model 176, Hardware Reference Manual, publication number 60456100.

The following are the basic rules for multiplying a floating point number.

1. Only work with positive operands.

   If operands are negative, complement all 60 bits.

   Save the original sign bits.

2. The results will be negative if the operands have unlike signs.

3. In this operation the exponents of the two operands are unpacked from the floating point format and are added with a correction factor of $+60_8$ to form the exponent for the Single Precision result.

   The coefficients are multiplied as sign integers to form a 96-bit integer product.

   The upper half of this product is then extracted to form the coefficient of the result.

4. When both operands are normalized, the result will be a normalized answer.

   To normalize, left shift the 96-bit integer product one place.

   Subtract one from the exponent.

5. If the two operands are not normalized, the upper half of the 96-bit product may be all zeros.

   This will result in a zero coefficient and a nonzero exponent.

6. A Partial Overflow occurs for this instruction whenever the exponent computation results in exactly $+1777_8$ and the result coefficient is taken from half of the 96-bit product.

7. A Complete Overflow occurs for this instruction whenever the exponent computation results in an exponent greater than $+1777_8$.

   This situation is sensed as a special case and a Complete Overflow word with proper sign, Overflow exponent and zero coefficient is delivered to the Xi Register using 3777 0--------0 or 4000 0--------0.

8. A Partial Underflow occurs for this instruction whenever the exponent computation results are exactly -1777_8 and the result coefficient is taken from the upper half of the 96-bit product.

9. A Complete Underflow occurs for this instruction whenever the exponent computation results in an exponent less than -1777_8.

   This situation is sensed as a special case and a complete zero word is delivered to the Xi Register using 0000 0---------0.

The following are examples of the Floating Product instruction.

- Xj = 1717 4000 0000 0000 0000
  Xk = 1717 4000 0000 0000 0000

  Xj = 1777 Xk = 1777
  1717 1717
  0060 -0060

  ∧ 4000 0000 0000 0000
  ∧ 4000 0000 0000 0000
  2000 0000 0000 0000 ∧ 0000 0000 0000 0000

  Rule 4 applies: if both operands are normalized, left shift the 96-bit product one place and subtract one from the exponent.

  4000 0000 0000 0000 0000 0000 0000 0000

  Xj = -0060 New Exponent = 1777
  Xk = -0060 -0061
  CF = +0060 1716
  -0060
  Normalize = -0001
  -0061

  Xi = 1716 4000 0000 0000 0000

The preceding example shows that the multiplicand and multiplier have the radix/decimal point to the left of the most significant bit (MSB). This means adding 60_8 to the result of the sum of the two exponents, which is called the CF or Correction Factor. Both operands were normalized and the result was not. Left shift the 96 product one place and subtract one from the exponent.

This does not show all the intermediate multiplication steps showing all the zeros.

- Xj = 5761 1117 7777 7777 7777
  Xk = 6001 5557 7777 7777 7777

  Rule 1 applies if operands are negative complement all 60 bits.

  Xj = 2016 6660 0000 0000 0000
  Xk = 1776 2220 0000 0000 0000

  Xj = 2016 Xk = -1777
  -2000 1776
  +0016 -0001

```
                          ^ 6660 0000 0000 0000
                          ^ 2220 0000 0000 0000
-------------------------------------------------
  15 5400 0000 0000 ^ 0000 0000 0000 0000
 155 4000 0000 0000 ¦ 00
1554 0000 0000 0000 ¦ 0
-------------------------------------------------
1747 1400 0000 0000 ^ 0000 0000 0000 0000
```

  Xj = +0016 New exponent 2000
  Xk = -0001 +0075
  CF = +0060 +2075
  +0075

  Xi = 2075 1747 1400 0000 0000

In this example both operands were negative so both operands were complemented. Therefore the result will be positive.

FLOATING PRODUCT 40 INSTRUCTION EXERCISE

This problem tests your understanding of the Floating Product instruction.

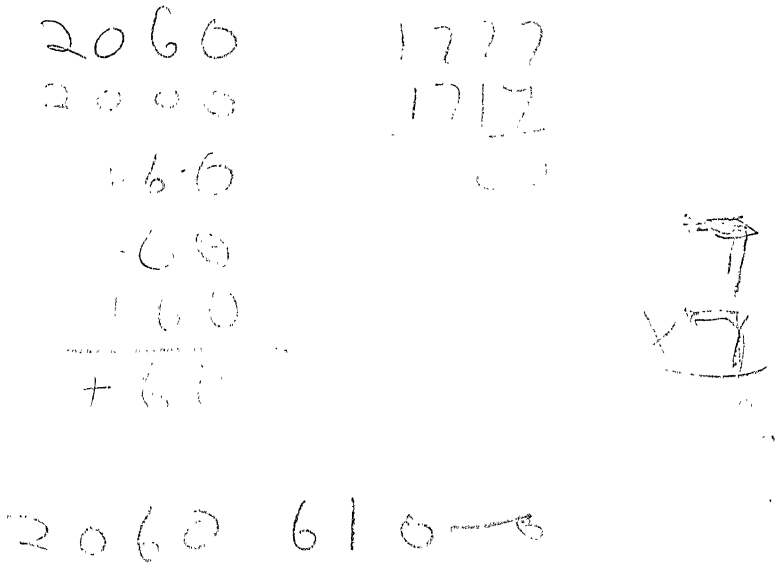
Directions: Work the following problem. Check your answer with that given on the following page.

1. Initial conditions:

   (X1) = 2060 7000 0000 0000 0000
   (X2) = 1717 7000 0000 0000 0000

   Location 10000 = 40312 46000 46000 46000

   What is the contents of the following register after execution of the above instruction?



X3 =

ANSWER FOR FLOATING PRODUCT INSTRUCTION EXERCISE

1. (X1) = 2060 7000 0000 0000 0000
   (X2) = 1717 7000 0000 0000 0000

```
X1 = 2060          X2 = -1777
    -2000               +1717
    +  60               -  60

                      ^ 7000 0000 0000 0000
                      ^ 7000 0000 0000 0000
-------------------------------------------
6100 0000 0000 0000 ^ 0000 0000 0000 0000

X1 = +0060      New Exponent = 2000
X2 = -0060                   +  60
      0000                    2060
CF = +0060
     +0060

X3 = 2060 6100 0000 0000 0000
```

In this example the result was positive since you multiplied like signs. The operands were normalized and in this case the answer was normalized so you did not have to left shift one place.

The 41 or Round Floating Product instruction is the same as Floating Product, except for the following.

The Round operation consists of adding a rounding bit to bit position 46 of the 96-bit product.

The following is an example.

- Xj = 2036 2222 2222 2222 2222
  Xk = 2064 0000 0000 0000 0022

```
Xj = 2036            Xk = 2064
     2000                 2000
   +   36               +   64

                                ^ 2222 2222 2222 2222
                                ^ 0000 0000 0000 0022
-------------------------------------------------------
                                ^ 4444 4444 4444 4444
                              4 | 4444 4444 4444 444
                             00 | 0000 0000 0000 00
          0000 0000 0000 0000 |
-------------------------------------------------------
          0000 0000 0000 0005 ^ 1111 1111 1111 1104
                                2<-This is equal to bit 46
                                   being equal to a one
-------------------------------------------------------
          0000 0000 0000 0005 ^ 3111 1111 1111 1104

Xj = +0036          New Exponent = 2000
Xk = +0064                         0202
       0122                        2202
CF = +0060
       0202
```

Xi = 2202 0000 0000 0000 0005

CYBER 170 Model 750/760 Functional Units
Learning Activity 10-A

ROUND FLOATING PRODUCT 41 INSTRUCTION EXERCISE

This problem tests your understanding of the Round Floating Product instruction.

Directions: Work the following problem. Check your answer with that given on the following page.

1. Initial conditions

   (X1) = 6060 3337 7777 7777 7777
   (X2) = 2063 0000 0000 0000 4440

   Location 10000 = 41312 46000 46000 46000

   What is the contents of the following register after execution of the above instruction?

X3 =

The answer to the previous exercise is on the following page.

ANSWER FOR ROUND FLOATING PRODUCT INSTRUCTION EXERCISE

1. (X1) = 6060 3337 7777 7777 7777
   (X2) = 2063 0000 0000 0000 4440

   X1 = 6060 3337 7777 7777 7777

   Complement as operand is negative

   1717 4440 0000 0000 0000

   X1 = 1777 − 1717 = 60

   X2 = 2063 − 2000 = + 63

```
                                  ^ 4440 0000 0000 0000
                                  ^ 0000 0000 0000 4440
                                  ^ 0000 0000 0000 0000
                            22    2000 0000 0000 000
                           222    0000 0000 0000 00
                          2220    0000 0000 0000 0
                          0000    0000 0000 0000
      0000 0000 0000 0000
       000 0000 0000 2464 ^ 2000 0000 0000 0000
                          ^ 2 = Round bit
      0000 0000 0000 2464 ^ 4000 0000 0000 0000
```

   Complement because of unlike signs

   7777 7777 7777 5313 3777 7777 7777 7777

   X1 = −0060 New Exponent = 2000
   X2 = +0063 0063
   0003 2063
   CF = +0060
   +0063

   Complement because of unlike signs

   5714

   X3 = 5714 7777 7777 7777 5313

The 42 or Double-Precision Product instruction is the same as Floating Product, except for the following.

The double-precision instruction consists of taking the lower 48 bits of the 96-bit product for the result. Do not add a correction factor of 60_8 to the sum of the two exponents.

The following is an example.

- Xj = 2000 4560 0000 0000 0000
  Xk = 1717 4000 0000 0000 0000

  Xj = 2000 − 2000 = 0

  Xk = −1777 + 1717 = −60

  ∧4560 0000 0000 0000
  ∧4000 0000 0000 0000
  2252 0000 0000 0000 ∧0000 0000 0000 0000

  Both operands were normalized. Left shift one

  4524 0000 0000 0000 0000 0000 0000 0000

  Xj = 0000 New Exponent = 1777
  Xk = −0060 −0061
  −0060 1716
  Normalizing −0001
  −0061

  Xi = 1716 0000 0000 0000 0000

CYBER 170 Model 750/760 Functional Units
Learning Activity 10-A

DOUBLE PRECISION PRODUCT 42 INSTRUCTION EXERCISE

This problem tests your understanding of the Double-Precision Product instruction.

Directions: Work the following problem. Check your answer with that given on the following page.

1. Initial conditions:

   (X1) = 3000 6000 0000 0000 4226
   (X2) = 2010 6000 0000 0000 0016

   Location 10000 = 42312 46000 46000 46000

   What is the contents of the following register after execution of the above instruction?

X3 =

The answer to the previous exercise is on the following page.

ANSWER FOR DOUBLE PRECISION PRODUCT INSTRUCTION EXERCISE

1. (X1) = 3000 6000 0000 0000 4226
   (X2) = 3000 6000 0000 0000 0016

```
X1 = 3000          X2 = 2010
     2000               2000
    +1000             +   10
```

```
                        ^ 6000 0000 0000 4226
                        ^ 6000 0000 0000 0016
-----------------------------------------------
                    4   ^ 4000 0000 0003 1604
                    6   : 0000 0000 0004 226
                   00   : 0000 0000 0000 00
 4400 0000 0000 0000    : 0
-----------------------------------------------
 4400 0000 0000 0012   ^ 4000 0000 0010 4064
```

```
X1 = 1000          New Exponent = 2000
X2 = 0010                         1010
     1010                         3010
```

X3 = 3010 4000 0000 0010 4064

You have completed learning activity 10-A. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 10-B. TEXT READING: MULTIPLY UNIT PRIMARY BLOCK DIAGRAM

This activity follows the data bits and control flow used by the 40, 41, and 42 instructions through the Multiply Functional Unit primary block diagram.

OBJECTIVE

- You will be able to follow the data bits and control flow through the Multiply Unit primary block 1.0 diagrams.

Directions: Place microfiche GF60420300W number 57 (6 of 10) in your microfiche viewer and select coordinate B3/B4.

The Multiply instruction requires five clock periods to complete execution. The 40 and 41 instructions are identical except the 41 instruction adds a value of one to bit 2^{-2}, which is then added to the product/result. The 42 is double-precision so its product is taken from the lower 48 bits 2^{-1} through 2^{-48} of the 96-bit result. If both operands are normalized, the result is normalized.

Although it takes five clock periods to do a multiply, you may enter multiplication instructions into the unit every other clock period.

As in the pencil and paper method, the bit-by-bit products are arranged as shown below and then added to obtain the final product, Xi.

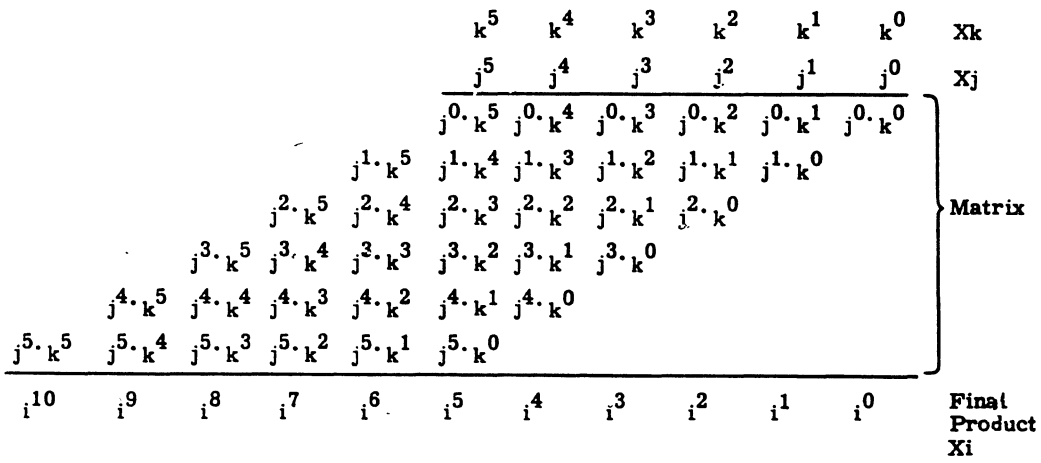

k^5 k^4 k^3 k^2 k^1 k^0 Xk
j^5 j^4 j^3 j^2 j^1 j^0 Xj
$j^0.k^5$ $j^0.k^4$ $j^0.k^3$ $j^0.k^2$ $j^0.k^1$ $j^0.k^0$
$j^1.k^5$ $j^1.k^4$ $j^1.k^3$ $j^1.k^2$ $j^1.k^1$ $j^1.k^0$
$j^2.k^5$ $j^2.k^4$ $j^2.k^3$ $j^2.k^2$ $j^2.k^1$ $j^2.k^0$
$j^3.k^5$ $j^3.k^4$ $j^3.k^3$ $j^3.k^2$ $j^3.k^1$ $j^3.k^0$
$j^4.k^5$ $j^4.k^4$ $j^4.k^3$ $j^4.k^2$ $j^4.k^1$ $j^4.k^0$
$j^5.k^5$ $j^5.k^4$ $j^5.k^3$ $j^5.k^2$ $j^5.k^1$ $j^5.k^0$
Matrix
i^{10} i^9 i^8 i^7 i^6 i^5 i^4 i^3 i^2 i^1 i^0
Final Product Xi

Figure 10-1. Multiplication Matrix

The Multiply is done in two steps. First, the lower 24 bits of Xj are multiplied to all 48 bits of Xk and the columns from this resulting matrix are partially added. Second, the upper 24 bits of Xj are multiplied to all 48 bits of Xk. The columns from this matrix are added to the Partial Sums and Partial Carries from the first Partial Add. This forms a 96-bit product/result.

INPUT REGISTERS

In quadrant A are the Xj and Xk holding registers for their exponents. If the value of the coefficient is negative, it will be complemented. In quadrant C is a holding register named Register No. 1. This holds both the Xj and Xk coefficients. These will be complemented if negative.

CONTROL

These diagrams show only one control. In quadrant C is Multiply Busy, which is the primary control signal in the Multiply Unit. This is set in the CPU at the end of the clock period in which the Multiply instruction was issued from the CIW Register. It is divided into three time periods, which control different sequences during the instruction execution. An example is that when the flag is set it blocks input to the Multiply Unit during the clock period immediately following instruction issue.

METHOD

Let's take the exponent first. In quadrant A are the two exponent registers which will hold Xj and Xk exponents which were received from the CPU. If they are negative, this data will be complemented. The $XJ+Xk+60_8$ Adder adds the two exponents during the second and third clock periods. This adder network for bits 0 through 3 and 6 through 10 is a two input adder. Bits 4 and 5, however, have three inputs, which allow you to add the 60_8 if single-precision.

Using the example from page 10-5, Xj = 1717 and Xk = 1717, the following example shows the adder input.

```
Bit          10 9   8 7 6   5 4 3   2 1 0

             Bits 0 through 9 were complemented to remove the bias

Xj           0 0    0 0 0    1 1 0    0 0 0
Xk           0 0    0 0 0    1 1 0    0 0 0
             0 0    0 0 0    0 0 0    0 0 0   Exclusive OR, or
60(8)                        1 1             Logical Difference
             0 0    0 0 0    1 1 0    0 0 0
             Complement bit 10 to add bias
              1 0    0 0 0    1 1 0    0 0 0
             Complement the output before sending to CPU
              0 1    1 1 1    0 0 1    1 1 1
                1        7        1        7     Resulting Exponent
```

This is what the result would have been for that example if the result had been normalized.

The Minus One network is for normalization correction. If both operands were normalized, the result must be normalized. If not, it is left shifted one place and one is subtracted from the exponent.

The exponent logic also checks the incoming exponents for special cases, such as Overflow, Underflow, or Indefinite before the addition. It also checks the result for Overflow, Underflow, or Indefinite and sets the appropriate special case flags. A Form Product Sign Adder receives the two sign bits. It performs a logical difference on the two bits. If like signs, the result is positive; unlike signs, the result will be negative.

The remaining logic in quadrants C and D is devoted to the multiplication of the coefficient. The Xk coefficient bits 0 through 47 are received and held by Register No. 1. The Xj coefficient splits into two halves. The lower half is bits 0 through 23 and the upper half is bits 24 through 47. These are also held in Register No. 1. Two passes complete the multiplication. First, bits 0 through 23 of Xj (lower half) are multiplied times bits 0 through 47 of Xk. These columns of the resulting matrix are partial added. Second, bits 24 through 47 of Xj (upper half) are multiplied times bits 0 through 47 of Xk. The bits from this resulting matrix are added to

the Partial Sums and Partial Carries (loop bits) from the first pass, which were right shifted 24_{10} places to position them properly. Figure 10-2 shows this two pass multiplication.

The multiply in the preceding example Figure 10-1 would appear as follows when executed in two steps.

First Pass					k^5	k^4	k^3	k^2	k^1	k^0	
								j^2	j^1	j^0	
					$j^0 \cdot k^5$	$j^0 \cdot k^4$	$j^0 \cdot k^3$	$j^0 \cdot k^2$	$j^0 \cdot k^1$	$j^0 \cdot k^0$	
				$j^1 \cdot k^5$	$j^1 \cdot k^4$	$j^1 \cdot k^3$	$j^1 \cdot k^2$	$j^1 \cdot k^1$	$j^1 \cdot k^0$		
			$j^2 \cdot k^5$	$j^2 \cdot k^4$	$j^2 \cdot k^3$	$j^2 \cdot k^2$	$j^2 \cdot k^1$	$j^2 \cdot k^0$			
Partial Sums			ps7	ps6	ps5	ps4	ps3	ps2	ps1	ps0	Results
Partial Carries			pc7	pc6	pc5	pc4	pc3	pc2			of First, Second and Third Level Adds
Second Pass		k^5	k^4	k^3	k^2	k^1	k^0				
					j^5	j^4	j^3				
		$j^3 \cdot k^5$	$j^3 \cdot k^4$	$j^3 \cdot k^3$	$j^3 \cdot k^2$	$j^3 \cdot k^1$	$j^3 \cdot k^0$				
	$j^4 \cdot k^5$	$j^4 \cdot k^4$	$j^4 \cdot k^3$	$j^4 \cdot k^2$	$j^4 \cdot k^1$	$j^4 \cdot k^0$					
$j^5 \cdot k^5$	$j^5 \cdot k^4$	$j^5 \cdot k^3$	$j^5 \cdot k^2$	$j^5 \cdot k^1$	$j^5 \cdot k^0$						
ps10	ps9	ps8	ps7	ps6	ps5	ps4	ps3				Results of First Level Add
pc10	pc9	pc8	pc7	pc6	pc5						
			ps7	ps6	ps5	ps4	ps3	ps2	ps1	ps0	Add Loop Bits from First Pass
			pc7	pc6	pc5	pc4	pc3	pc2			
ps10	ps9	ps8	ps7	ps6	ps5	ps4	ps3	ps2	ps1	ps0	Results of Second and Third Level Adds
pc10	pc9	pc8	pc7	pc6	pc5	pc4	pc3				
i^{11}	i^{10}	i^9	i^8	i^7	i^6	i^5	i^4	i^3	i^2	i^1	i^0 Final Add

In a 24 by 48 bit multiply, in order to handle 1152 bit products in the matrix, the summing is performed in several more stages and there are several more levels of sums and carries. Also, because of the size of the matrix, there are several sums or carries per bit position at some levels of the multiply. Since a complex carry network to resolve all the carries would require excessive time and hardware, the multiply unit employs a Carry Save Adder for efficiency.

Figure 10-2. Multiplication Matrix for Two Passes

CARRY SAVE ADDER

The Carry Save Adder used in the Floating Multiply Unit permits the addition of columns of numbers without using the time-consuming carry and satisfy checks typical of full adders. Instead, each level of add has the following outputs, Pseudo Sums and Pseudo Carries.

	1	0	1	0
	0	1	1	0
Pseudo Sums	1	1	0	0
	1	0	1	0
	0	1	1	0
Pseudo Carries	0	0	1	0

(Pseudo Carries are displaced one bit to the left, because they affect the next significant bit position.)

The sum of these two numbers is the actual answer; however, the sum is not taken until the Final Add. Instead, both quantities are put back into the next level of the adder and are summed with other Pseudo Sum and Pseudo Carry bits. These bits (the sums of bits in the same column of the matrix or carries to that column) represent the same bit position of the final product.

Thus, the inputs to the First-Level Add are the logical products produced by the matrix. The inputs to the Second-Level Add are the Pseudo Sums and Pseudo Carries from the First-Level Add, and the inputs to the Third-Level Add are the Pseudo Sums and Pseudo Carries from the Second-Level Add. The output from the Third-Level Add is a Pseudo Sum and Pseudo Carry bit for each bit position of the product.

After the first pass, the output from the Third-Level Add is shifted right 24 places, fed back into the Second-Level Add, and merged with the results of the second pass through the matrix. The Second-Level and Third-Level Adds are then repeated for the entire product until a Pseudo Sum and a Pseudo Carry bit again represent each bit position of the product. The output of the Third-Level Add is then fed to the Final Add network and reduced to one bit for each bit position of the final product (96 bits).

The final logic is the Normalize logic. If both operands were normalized, this logic will left shift the result one place, before sending it to the CPU. If rounding, bit 2^{-2} will be added to the result at this time. Read notes (1) and (2).

You have completed learning activity 10-B. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 10-C. TEXT READING: MULTIPLY UNIT DETAILED PAK DIAGRAMS

This activity follows the data bits and control flow through the Multiply Unit detailed 3.0 diagrams.

OBJECTIVE

- You will be able to follow the data bits and control signals through the Multiply Unit detailed 3.0 diagrams.

Directions: Place microfiche GF60420300W number 57 (3 of 10) in your microfiche viewer and select coordinate D1/D1.

The 3.5 diagram is for the exponent, special case, and control.

INPUT REGISTERS

In quadrant A and C is a KR module at 7F16 that has the Xj and Xk Exponent Registers. These receive bits 48 through 59 when gated by Multiply Busy. If the operand is negative, this exponent is complemented.

CONTROL

The Control logic is found in quadrant B on the KV module at 6H14. The Multiply Busy Flag is received from the CPU at the end of the first clock period that the instruction was issued from the CIW Register. This means that Multiply Busy will be equal to one only during the second clock period. This and Multiply Busy signals then start timing chains that control the sequence of data through the unit. The following are most of the control signals and their functions.

ROUND BIT	Causes a round operation to be performed during the third clock period if specified by the instruction.
GO MULTIPLY SPECIAL CASE	Causes a special case flag to be sent to the CPU during the fourth clock period if a special case exists.

GATE LOWER HALF	Gates coefficient bits 0 through 47 or shifted coefficient bits -1 through 46 to the CPU during the fifth clock period.
GATE UPPER HALF	Gates coefficient bits 48 through 95 or shifted coefficient bits 47 through 94 to the CPU during the fifth clock period.
GATE EXPONENT	Gates the exponent to the CPU during the fifth clock period.
ENABLE LOOP	Gates the loop bits from the first pass during the third clock period.

The remaining diagrams cover these controls.

ADD NETWORK

Taking the two sign bits from the Input Registers first, these go to the BA module at 7H16. The Form Product Sign logic performs an Exclusive OR or Logical Difference on the two sign bits with the following results.

0	1	1	0
$\overline{1}$	$\overline{0}$	$\overline{1}$	$\overline{0}$
$\overline{1}$	$\overline{1}$	$\overline{0}$	$\overline{0}$

In multiplication, like signs give you a positive result and unlike signs give a negative result. This result then goes to the KV module at 7H14. At CP02 or clock three, it is gated to the Register MB+2. At clock four it is sent to the CPU X Sign Control.

Going back to the Input Registers on the KR module, the two exponents, bits 48 through 58, from this point on will be labeled bits 0 through 10. During the second clock period, the exponent is in the KS module at 7G16. Bits 0 through 9 are complemented. Exponent bias is removed by not complementing bit 10. Using the example on page 10-19, the inputs to the adder Xj and Xk would equal 0060. If single-precision then 60_8 would be added also. This value is controlled by DP Corr. (Double-Precision Correction) from the KV module. Input m1 equals one on a 42 or DP Instruction with Multiply Busy equal to a one during clock one. This sets m1 FF (Flip FLop) and DP Corr. This signal causes the value of 00 to be added to Xj and $\overline{\text{Xk}}$.

First Half Add

The First Half Add Network forms a Partial Sum for bits 0 through 3 and bits 6 through 10 by adding those bits of $\overline{Xj}$ and $\overline{Xk}$. Bits 4 and 5 of $\overline{Xj}$, of $\overline{Xk}$ and of 60_8 are also added by a three input adder. These Sums and Carries go to the Second Half Add.

Second Half Add

The Second Half Add, which is on the BA module at 7H16, adds the Carries and Sums. In our example, this would be 3717 as its output is always complement data. This value is sent to the $\overline{EXP}$ Result Register on the KU module at 7H15 during the fourth clock period. This result is used by two separate output networks taking the top logic first. Add bias complements bit 10, which would make the example equal to 1717. This goes to the AND NOT function, where the following gates are necessary.

Top gate is GATE EXPONENT (CP04), which takes place during clock five. Bottom gate is $\overline{\text{Left Shift}}$. If both operands were not normalized you do not need to left shift, so $\overline{\text{Left Shift}}$ would equal one. The functions output would equal 0060. The sign bit is a logical one so the output to the CPU would equal 4060. The actual sign bit that is stored in the Xi Register is determined by X Register Input Control in the CPU.

The bottom logic has an $\overline{EXP}$+1 adder, which would result in subtracting one from the exponent (see note (2)). In this case, the output would equal 3720 through the AND NOT function. Again you would have GATE EXPONENT, but because the operands were normalized and the product was not, left shift would equal one. The functions output would equal 0057. The sign bit is the complement of GATE EXPONENT, so this would equal zero. The output to the CPU would equal 0057. This completes a normal exponents formation.

Special Case

The KS module at 7G16 checks for Overflow, Underflow, or Indefinite in the incoming exponents. If any are detected, it sends a signal to the Check for Special Case logic on the BA module at 7H16. This logic also checks the output of the Second Half Add for special cases. Figure 10-3 shows the resulting flags sent to the CPU when various combinations are used. This is also found in Section Five of the hardware reference manual. The designations W and N are defined as follows:

W any word except ± ∞ and ± IND

N any work except ± ∞ and ± IND and +0

		Xk						
		+N	-N	+0	-0	+∞	-∞	+ IND
Xj	+N			0	0	+∞	-∞	IND
	-N			0	0	-∞	+∞	IND
	+0	0	0	Integer † multiply		IND	IND	IND
	-0	-0	0			IND	IND	IND
	+∞	+∞	-∞	IND	IND	+∞	-∞	IND
	-∞	-∞	+∞	IND	IND	-∞	+∞	IND
	± IND	IND	IND	IND	IND	IND	IND	IND

† If both operands used in the integer multiply are normalized, an underflow results.

Figure 10-3. Xj Multiplied by Xk (40, 41, 42 Instructions)

The remaining logic is on the BA module at 7H16. The check for Integer Multiply logic is in quadrant C. An Integer Multiply result is an exponent of all zeros. This will take place if both operands have Underflow (-1777 unbiased) exponents and both coefficients are not normalized.

NOTE

If an Integer Multiply is executed with one coefficient normalized, an undetected Overflow result will normally occur.

The logic looks at Underflow from the KS module and at bit 94 from the GC module on the 3.0 diagram. If this is equal to a zero, an Integer Multiply is performed.

This Integer Multiply to the Second Half Add forces bits 0, 4, and 5 to logical ones, and bit 10 to a logical zero. This assures that all zero bits will be sent to the Xi Register for the exponent portion. If the sign of the result was negative, the complement will be stored in the Xi Register. This completes the formation of the exponent.

If any portion of this material is unclear, review the exponent diagrams now. If not, select coordinate D1/D2.

COEFFICIENT FORMATION

Coefficient Multiply

Coefficient multiplication is accomplished in two passes through the logic as shown in figure 10-2. First, bits 0 through 47 of Xk are multiplied by bits 0 through 23 of Xj. The columns of the resulting matrix are partially added, resulting in Partial Sums and Carries.

Second, bits 0 through 47 of Xk are multiplied by bits 24 through 47 of Xj. The columns of this resulting matrix are partially added, resulting in Partial Sums and Carries.

The resulting Partial Sums and Carries (loop bits) from the first multiplication (first pass) are right shifted 24_{10} places. The Partial Sums and Carries from the two passes are then added for the final result.

Let's use a simple example of this, 22 times 11.

$$\begin{array}{r} 22 \\ \underline{11} \\ 22 \\ \underline{22} \\ 242 \end{array}$$

Use figure 10-4 for computing the Pseudo Sums and Carries.

	1	0	1	0		1	0	1	0
	0	1	1	0		0	1	1	0
Pseudo Sums	1	1	0	0	Pseudo Carries	0	0	1	0

(Pseudo Carries are moved one bit position to the left, because they affect the next significant bit position.)

Figure 10-4. Pseudo Sums and Pseudo Carries

First Pass	Xk		010	010	
	Xj			001	
			010	010	
		0	000	00	Bit by Bit
		00	000	0	Multiplication
1st Pass Partial Sums		00	010	010	
1st Pass Partial Carries		00	000	0	

Second Pass	Xk		010	010	
	Xj			001	
			010	010	
		0	000	00	Bit by Bit
		00	000	0	Multiplication
2nd Pass Partial Sums		00	010	010	
2nd Pass Partial Carries		00	000	0	

2nd Pass Partial Sums	00	010	010	
2nd Pass Partial Carries	00	000	0	
1st Pass Partial Sums Right Shifted		00	010	010
1st Pass Partial Carries Right Shifted		00	000	0
Partial Sums	00	010	000	010
Partial Carries	00	000	100	0
Final Add	00	010	100	010
		2	4	2

The entire multiplication process can be separated into the following sections.

Holding Register and Fanouts	GA Modules
Matrix and First-Level Add	GB and GC Modules
Holding Register & Second-Level Add	GD, GE, GF, GG, GH, GI, and GJ Modules
Third-Level Add	GK, GL, GM, GN, GO, and GP Modules
Holding Register and Final Add, First Half	GQ, GR, GS, and GT Modules
Final Add, Second Half	GU, GV, GW, and GX Modules

INPUT REGISTERS

The input registers are on the GA modules in quadrant A. Using 7E13 for an example, notice the bits it holds. For Xj it holds bits 0, 1, 24, and 25 and for the Xk operand, it holds bits 0 through 3. These modules receive bits 0 through 47 of the coefficient and the sign bit, bit 59. If the sign is negative, equal to a one, that coefficient is complemented. Recalling from the Control logic, that Multiply Busy was not set until the second clock period. Multiply Busy will be equal to a one while loading the input registers. This storing is taking place during the same clock period that the Multiply instruction was issued from CIW Register. When Multiply Busy goes set during the second clock, Multiply Busy becomes a zero and blocks any entry of new data. The data then in the input registers is held for two clock periods.

Xk Register bits 0 through 3 go to a fanout, which sends those bits to the GB and GC modules. Xj bits 0 and 1 go to a Fanout and then go to the GB and GC modules. This would be the first pass when you multiply Xk bits 0 through 47 by Xj bits 0 through 23. Multiply Busy goes set on the second clock period. This makes Multiply Busy equal to a zero. MBFF (Multiply Busy Flip Flop) will equal a one making the AND function gate bits 24 and 25 to the Xj Holding Register 0 and 1 or 24 and 25. Now the unit can start multiplying its second pass which means multiplying Xk bits 0 through 47 by Xj bits 24 through 47, for forming the second half of the product.

The following shows that there are a total of 10 Xi bit 9s and how they are formed by multiplying 10 bits by 10 bits.

Bits	12	11	10	9	8	7	6	5	4	3	2	1	0
Xk				0	0	0	0	0	0	0	0	0	0
Xj				0	0	0	0	0	0	0	0	0	0
				0•9	0•8	0•7	0•6	0•5	0•4	0•3	0•2	0•1	0•0
			1•9	1•8	1•7	1•6	1•5	1•4	1•3	1•2	1•1	1•0	
		2•9	2•8	2•7	2•6	2•5	2•4	2•3	2•2	2•1	2•0		
	3•9	3•8	3•7	3•6	3•5	3•4	3•3	3•2	3•1	3•0			
	4•8	4•7	4•6	4•5	4•4	4•3	4•2	4•1	4•0				
	5•7	5•6	5•5	5•4	5•3	5•2	5•1	5•0					
	6•6	6•5	6•4	6•3	6•2	6•1	6•0						
	7•5	7•4	7•3	7•2	7•1	7•0							
	8•4	8•3	8•2	8•1	8•0								
	9•3	9•2	9•1	9•0									
Result Bits	12	11	10	9	8	7	6	5	4	3	2	1	0

Using this information let us look at the matrix and first level add.

MATRIX AND FIRST LEVEL ADD

These 24 GB and GC modules form the bit-by-bit logical product of Xk bits 0 through 47 and Xj bits 0 through 23 during the second clock period (first pass). During the third clock period (second pass) Xk bits 0 through 47 and Xj bits 24 through 47 form their bit-by-bit logical products. Using the GC module at 7B06 in quadrant D, note the different bits it handles: Xj bits 0 through 7 during the first pass, Xj bits 24 through 31 on the second pass, and Xk bits 0 through 7. It also handles result bits (Xi bits) 0 through 10 on the first pass and bits 24 through 34 on the second pass. It illustrates how several bits can be the same bit position in the product. For example, Xi product bit 9 is formed from the following operand bit combinations, starting with the top AND function.

```
Xj bit 2 . Xk bit 7 = Xi bit 9      First level Add = Pseudo sum
Xj bit 3 . Xk bit 6 = Xi bit 9      bit 9 and Pseudo Carry bit 10

Xj bit 6 . Xk bit 3 = Xi bit 9      First level Add = Pseudo sum
Xj bit 7 . Xk bit 2 = Xi bit 9      bit 9 and Pseudo Carry bit 10
```

The First-Level Add sums these product bits in groups of two or three bits to produce Pseudo Sum and Pseudo Carry bits. As the previous example shows, the First-Level Add of the top two Xi bit 9s produced a Pseudo Sum bit 9 and a Pseudo Carry bit 10. The sum of the lower two Xi bit 9s produced a Pseudo Sum bit 9 and a Pseudo Carry bit 10. Thus, four bits are output by 6B06 for Xi bit 9 (two Pseudo Sums and two Pseudo Carry bits). You can see that there are four outputs by looking at the N+9 output on 7B06, which shows four outputs. Thus the GC module has taken care of four of the possible 10 combinations needed to form result bit nine.

To form the other six combinations look at the GB module at 7A04 in quadrant D. On this module, Xi product bit 9 is formed from the following operand bit combinations, starting with the top AND function.

Xj bit 0 . Xk bit 9 = Xi bit 9
Xj bit 1 . Xk bit 8 - Xi bit 9
Xj bit 4 . Xk bit 5 = Xi bit 9
Xj bit 5 . Xk bit 4 - Xi bit 9
Xj bit 8 . Xk bit 1 = Xi bit 9
Xj bit 9 . Xk bit 0 - Xi bit 9

Bit 9 on this module is N+1. Looking at the output of N+1 you will see it is outputting a total of only three Pseudo Sums and Pseudo Carries.

Figure 10-5, Sample logic modules in First Level Add, shows three of the modules used in First-Level Add. The right side shows the GC module at 7B06. The uppermost diagram under GC and second column from left shows four bits are inputted. These would be bits 2, 3, 6, and 7. The term G5 is a matrix term output and is the AND function of bits 7·2. Your detailed pak diagrams show this. The bottom diagram shows the Xi or resulting function of bits and the two Pseudo Sums and two Pseudo Carries from that module.

Take the center diagrams, which show the GB Module at 7A04. The uppermost diagram under GB and the second column from the right shows six bits are inputted. These would be bits 0, 1, 4, 5, 8, and 9. The term A1 is a matrix term output and is the AND function of bits 9·0. The bottom diagram again shows the Result bit in this case two Pseudo Sums and one Pseudo Carry. This then shows that for result bit 9 First-Level Add you used the GB module at 7B06 and the GC module at 7A04.

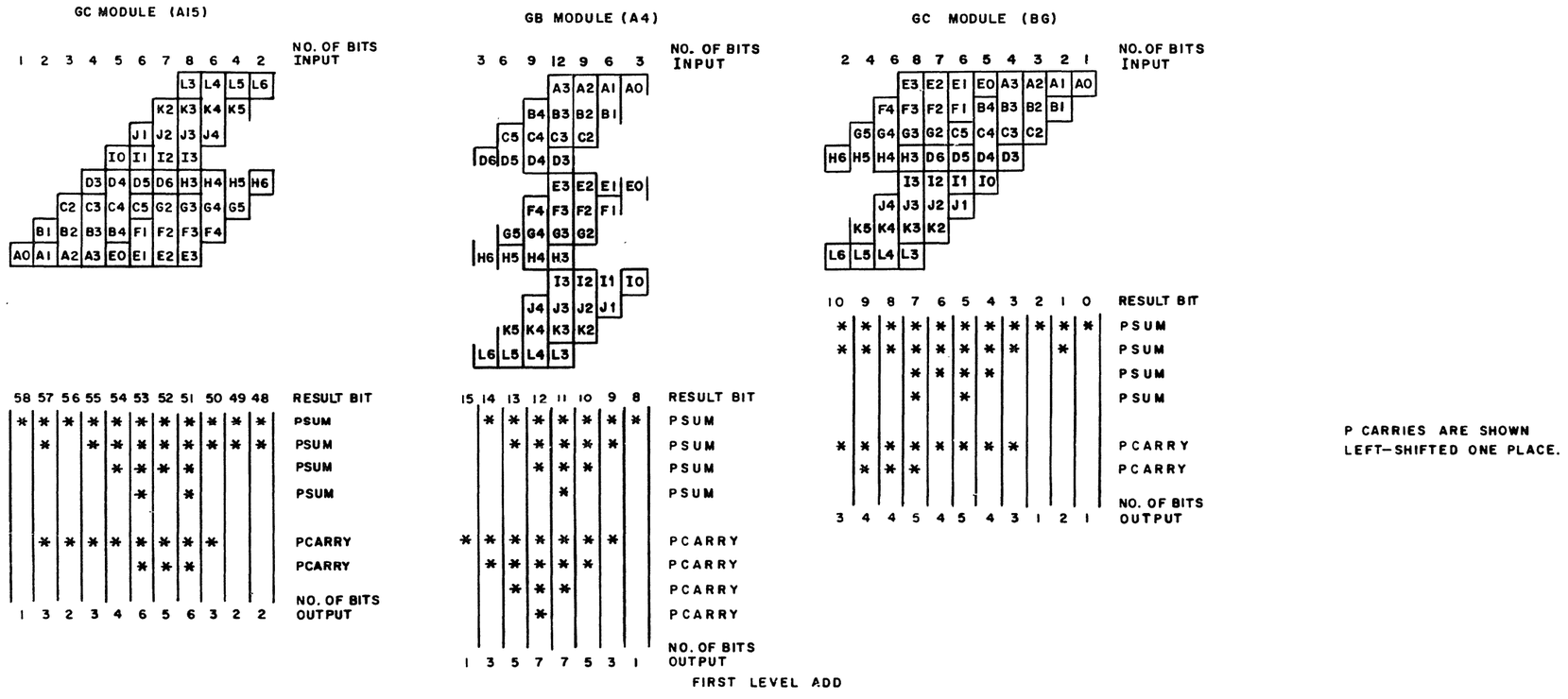

GC MODULE (A15)
1 2 3 4 5 6 7 8 6 4 2
NO. OF BITS INPUT
L3 L4 L5 L6
K2 K3 K4 K5
J1 J2 J3 J4
I0 I1 I2 I3
D3 D4 D5 D6 H3 H4 H5 H6
C2 C3 C4 C5 G2 G3 G4 G5
B1 B2 B3 B4 F1 F2 F3 F4
A0 A1 A2 A3 E0 E1 E2 E3
58 57 56 55 54 53 52 51 50 49 48
RESULT BIT
PSUM
PSUM
PSUM
PSUM
PCARRY
PCARRY
1 3 2 3 4 6 5 6 3 2 2
NO. OF BITS OUTPUT
GB MODULE (A4)
3 6 9 12 9 6 3
NO. OF BITS INPUT
A3 A2 A1 A0
B4 B3 B2 B1
C5 C4 C3 C2
D6 D5 D4 D3
E3 E2 E1 E0
F4 F3 F2 F1
G5 G4 G3 G2
H6 H5 H4 H3
I3 I2 I1 I0
J4 J3 J2 J1
K5 K4 K3 K2
L6 L5 L4 L3
15 14 13 12 11 10 9 8
RESULT BIT
PSUM
PSUM
PSUM
PSUM
PCARRY
PCARRY
PCARRY
PCARRY
1 3 5 7 7 5 3 1
NO. OF BITS OUTPUT
GC MODULE (BG)
2 4 6 8 7 6 5 4 3 2 1
NO. OF BITS INPUT
E3 E2 E1 E0 A3 A2 A1 A0
F4 F3 F2 F1 B4 B3 B2 B1
G5 G4 G3 G2 C5 C4 C3 C2
H6 H5 H4 H3 D6 D5 D4 D3
I3 I2 I1 I0
J4 J3 J2 J1
K5 K4 K3 K2
L6 L5 L4 L3
10 9 8 7 6 5 4 3 2 1 0
RESULT BIT
PSUM
PSUM
PSUM
PSUM
PCARRY
PCARRY
3 4 4 5 4 5 4 3 1 2 1
NO. OF BITS OUTPUT
P CARRIES ARE SHOWN LEFT-SHIFTED ONE PLACE.
FIRST LEVEL ADD

Figure 10-5. Sample Logic Modules in First Level Add

Figure 10-6 shows one half of the Multiply. Look at Xi bit 9 and the 10 matrix terms under it. These agree with the terms in figure 10-5.

There are 788 Pseudo Sums and Carries generated by First-Level Add each pass. These bits are sent to Second-Level Add on the 3.1 diagram at coordinate D3/D4.

The following block diagram in figure 10-7 and figure 10-8 are aids for the following discussion.

SECOND-LEVEL ADD (First Pass Only)

The Second-Level Add modules in coordinate D9/D10 receive 788 Pseudo Sums and Carries from First-Level Add. This logic continues to sum groups of these bits to form Pseudo Sum and Carry bits for each bit position of the result. These resultant bits are held in registers for one clock period until the bits of the First-Level Add, second pass, arrive.

During the first pass (second clock) these Pseudo Sums and Carries are summed on modules GD, GE, GF, GG, GH, and GI. These bits are summed into groups to form Pseudo Sum and Pseudo Carry bits for each bit position of the lower portion of the product (bits 0 through 70). These bits are stored in the register where they are held for use during the third clock period.

At the beginning of the third clock period, these bits 0 through 70 go through a second adder on the same modules. This further sums these Pseudo Sums and Carries before sending them to the Third-Level Adder, first pass.

The GJ modules are not used during the first pass.

THIRD-LEVEL ADD (First Pass Only)

The Third-Level Add modules on coordinate E1/E2 receives 336 Pseudo Sums and Carries from Second-Level, first pass. This adder continues to sum groups of these bits to form Pseudo Sums and Carry bits for each bit position of the result. The GK, GL, GO, and GP modules have registers that hold some of the Pseudo Sums, Pseudo Carries, and Loop Bits. On the GM, GN, GO, and GP modules, reference is made to duplicate bits. Read note ④ also. These bits are received from fanouts (not shown) on the Second-Level Add modules. The duplicate bits generate a Pseudo Carry to bit N from bit N-1 and bit N-2. The Pseudo Sums for bit N-1 are generated on the next lower modules as bit N+2 Pseudo Sums. Since separate modules generate Pseudo Sums and Carries, duplication of input bits is necessary.

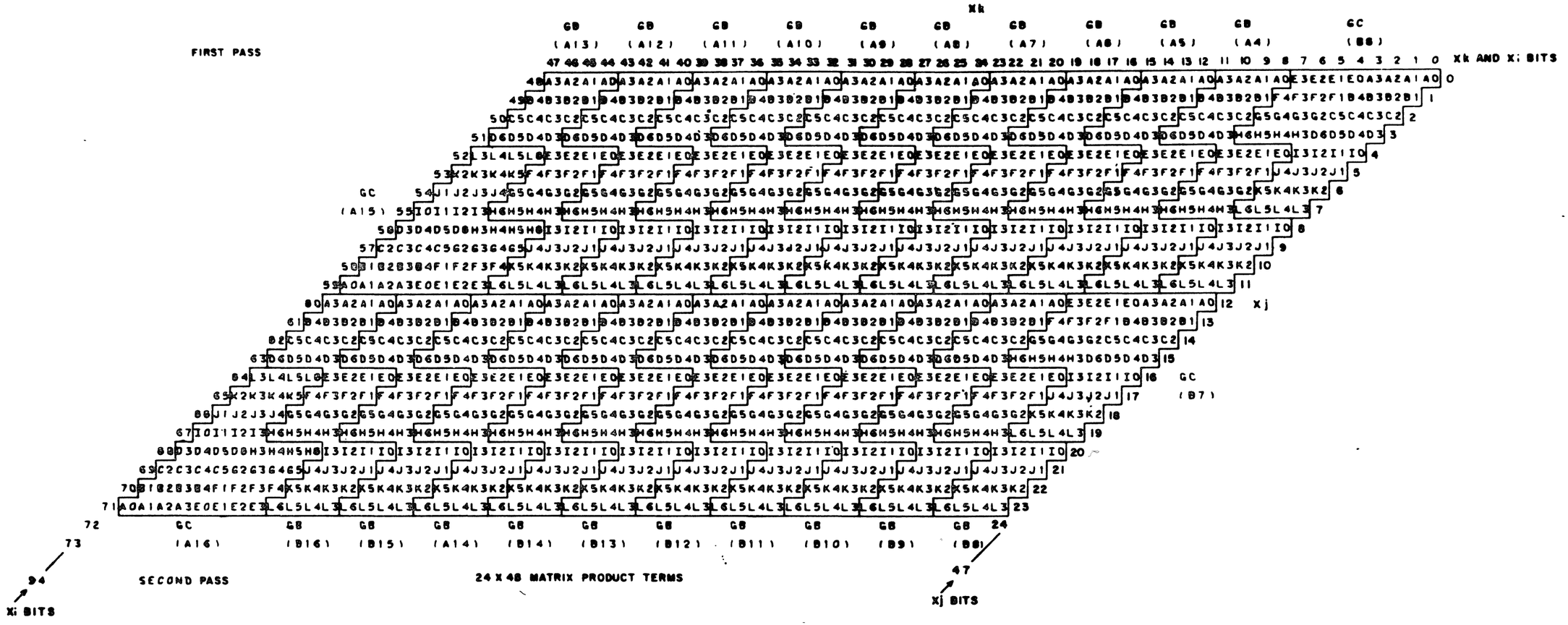


Figure 10-6. Multiply Unit 24 by 48 Matrix (Full Grid)

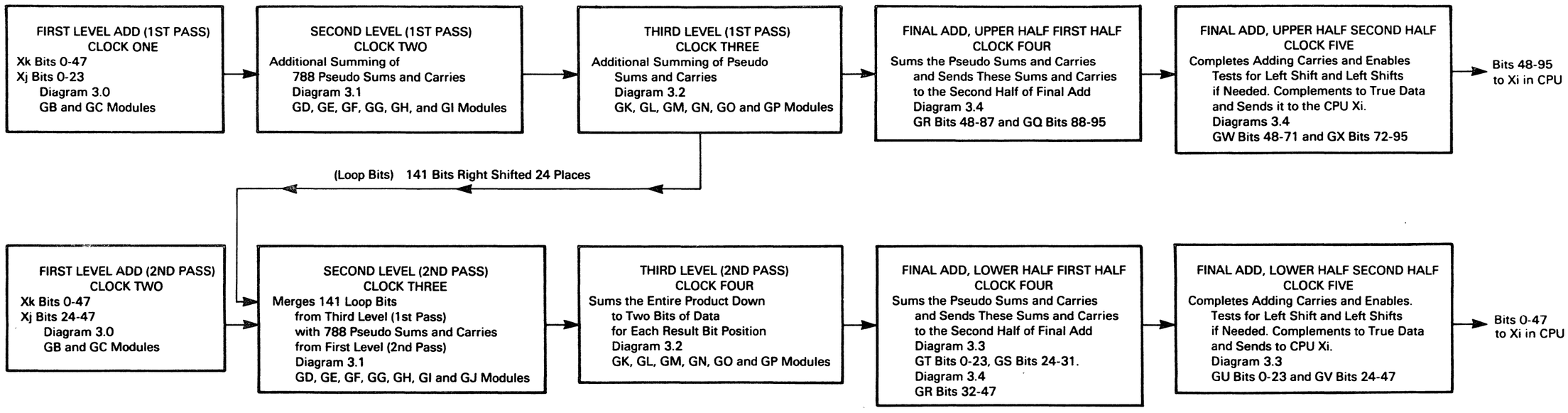
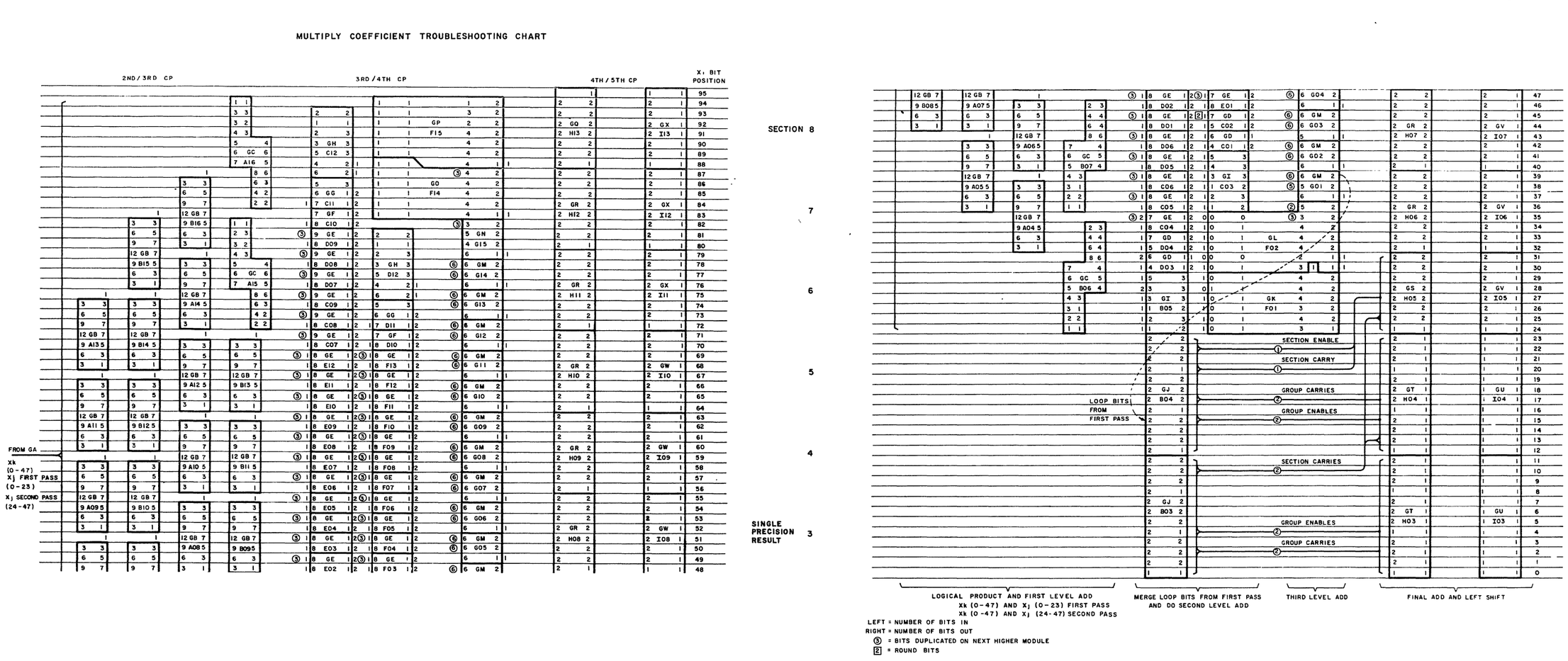

FIRST LEVEL ADD (1ST PASS)
CLOCK ONE
Xk Bits 0-47
Xj Bits 0-23
Diagram 3.0
GB and GC Modules
SECOND LEVEL (1ST PASS)
CLOCK TWO
Additional Summing of
788 Pseudo Sums and Carries
Diagram 3.1
GD, GE, GF, GG, GH, and GI Modules
THIRD LEVEL (1ST PASS)
CLOCK THREE
Additional Summing of Pseudo
Sums and Carries
Diagram 3.2
GK, GL, GM, GN, GO and GP Modules
FINAL ADD, UPPER HALF FIRST HALF
CLOCK FOUR
Sums the Pseudo Sums and Carries
and Sends These Sums and Carries
to the Second Half of Final Add
Diagram 3.4
GR Bits 48-87 and GQ Bits 88-95
FINAL ADD, UPPER HALF SECOND HALF
CLOCK FIVE
Completes Adding Carries and Enables
Tests for Left Shift and Left Shifts
if Needed. Complements to True Data
and Sends it to the CPU Xi.
Diagrams 3.4
GW Bits 48-71 and GX Bits 72-95
Bits 48-95
to Xi in CPU
(Loop Bits) 141 Bits Right Shifted 24 Places
FIRST LEVEL ADD (2ND PASS)
CLOCK TWO
Xk Bits 0-47
Xj Bits 24-47
Diagram 3.0
GB and GC Modules
SECOND LEVEL (2ND PASS)
CLOCK THREE
Merges 141 Loop Bits
from Third Level (1st Pass)
with 788 Pseudo Sums and Carries
from First Level (2nd Pass)
Diagram 3.1
GD, GE, GF, GG, GH, GI and GJ Modules
THIRD LEVEL (2ND PASS)
CLOCK FOUR
Sums the Entire Product Down
to Two Bits of Data
for Each Result Bit Position
Diagram 3.2
GK, GL, GM, GN, GO and GP Modules
FINAL ADD, LOWER HALF FIRST HALF
CLOCK FOUR
Sums the Pseudo Sums and Carries
and Sends These Sums and Carries
to the Second Half of Final Add
Diagram 3.3
GT Bits 0-23, GS Bits 24-31.
Diagram 3.4
GR Bits 32-47
FINAL ADD, LOWER HALF SECOND HALF
CLOCK FIVE
Completes Adding Carries and Enables.
Tests for Left Shift and Left Shifts
if Needed. Complements to True Data
and Sends to CPU Xi.
Diagram 3.3
GU Bits 0-23 and GV Bits 24-47
Bits 0-47
to Xi in CPU

Figure 10-7. Flow Chart of Coefficient Multiplication



Upper Half

Lower Half

Figure 10-8. Multiply Coefficient Troubleshooting Chart

First Pass

The first pass through these modules takes place during the third clock period. This logic continues to sum the results of Pseudo Sums and Carries down to a Pseudo Sum and Pseudo Carry (Loop Bits) for each bit position of the lower part of the product. These 141 bits loop back to the Second-Level Add and are added to the results of the First-Level Add, second pass. This is the same as the second pass through the matrix.

SECOND LEVEL ADD (Second Pass Only) Coordinates D9/D10

Second Pass

The second pass uses all the modules including the two GJ modules that are for bits 0 through 23. This add takes place during the third clock period. This logic is adding the 141 Loop Bits (bits 0 through 70) from Third-Level Add, first pass, to the 788 Pseudo Sums and Carries from the matrix or First-Level Add, second pass. The results of this add are held in the registers on each of the modules until the fourth clock period. At the beginning of the fourth clock period, these modules will further sum the Pseudo Sums and Carries before they are sent to the Third-Level Add, second pass.

THIRD LEVEL ADD (Second Pass Only) Coordinates E1/E2

Second Pass

The second pass takes place during the fourth clock period. At this time, the Third-Level Adder further sums the entire product (the result of both passes through the matrix) until two bits of data remain for each bit position (bits 24 through 95). These Pseudo Sums are Carries. These bits are sent to the Final Add logic.

FINAL ADD, LOWER HALF

The Final Add is done in two parts (halves). The first half sums the Pseudo Sums and Carries and sends the resulting Carries and Enables to the second half of the Final Add. The second half uses a standard pass and carry check method of carry propagation. Both halves of the Final Add are a full adder. A full adder was not used in the earlier stages because of the extra time involved in checking Enables and Carries.

FIRST HALF

The first half of the Final Add takes place during the fourth clock period. The logic is located on two sets of diagrams. Using the 3.3 diagrams on coordinate E2/E3 and the GT modules in quadrant C, the logic is receiving Section Carries, Group Carries and Group Enables from the GJ modules. These are added forming result bits 0 through 23. These are held in a register until clock five. The GS modules in quadrant A receive Pseudo Sums and Carries or two bits of data for each resulting bit. The result from Begin Final Add is held in the register until clock five.

Select coordinate F1/F2 and the two GR modules in quadrant C, locations 7H06 and 7H07. These are receiving Pseudo Sums and Carries or two bits of data for each resulting bit. This result from the Begin Final Add is also held in the register until clock five.

SECOND HALF

The second half of the Final Add takes place during the fifth clock period. The logic again is on two sets of diagrams. Using the 3.4 diagrams on coordinate F1/F2, partial result is stored in a register on two GR modules at 7H06 and 7H07 from clock four. These bits then go through the Final Add and go to the GV modules on the 3.3 diagrams at coordinate E7/E8. These modules are located in quadrant B at location 7I06 and 7I07. These will be result bits 32 through 47. In quadrant A a Partial Sum was in the register. This goes through Final Add to a GV module to the Complete Final Add logic. This output is sent to an AND NOT function which has two other gating commands. The bottom is GATE LOWER HALF. The top is LEFT SHIFT. As previously noted, if both coefficients are normalized, and the result is not, it must be left shifted one place. Assume for now that this is not the case. LEFT SHIFT would equal a one. With all gates made, the result of the adder would be complemented giving you true data. These three modules would send bits 24 through 47 to the Xi Register in the CPU.

Assume that the coefficient had to be left shifted one place. The output would have gone through the Left Shift 1 logic. Test for Left Shift would cause LEFT SHIFT to equal a one. With GATE LOWER HALF equal to a one you would again complement through the AND NOT function. You would be outputting bits 25 through 48 to the Xi Register. LEFT SHIFT would equal a zero so there would be no output from that function.

The remaining 24 bits or bits 0 through 23 leave the registers on the two GT modules in quadrant C on clock five to the two GU modules. These 24 bits are in true form. They are complemented so all bits prior to output are in complement form. As seen on the GV modules, the output can be left shifted one and all pass through AND NOT functions as they are sent to the Xi Register in the CPU. This completes the lower half. The upper half is like bits 24 through 47 of the lower half.

FINAL ADD, UPPER HALF

The logic is on the 3.4 diagrams on coordinate F1/F2.

FIRST HALF

During clock four there are two bits for each result bit in Xi. In quadrant C are five additional GR modules which will do a Partial Add and hold that result until clock five. In quadrant A is a GQ module, which handles the remaining eight bits.

Note Pseudo Sum 94. This bit is the logical product of bit 47 of your two source operands. If both operands had been normalized, this bit should be equal to a one. If it is not equal to a one, it indicates your two source operands were not normalized. Let's assume the operands were normalized. This Pseudo Sum 94 is used to set up the Test for Left Shift logic circuits on the KU, GU, GV, GW, and GX modules. Using the GX module at 7I13 in quadrant B, the Pseudo Sum 94 bit would have been used as a Section Carry bit to the Test for Left Shift logic, which looks to see if bit 95 is set. If bit 95 is not set, then the logic will cause the product to be left shifted one bit position. An example of this can be seen in learning activity 10-A.

SECOND HALF

The second half of the Final Add takes place during the fifth clock period and is accomplished on the GW and GX modules. This logic is the same as the output of Lower Half logic, with the exception of one gating term. The output uses GATE UPPER HALF instead of GATE LOWER HALF. This completes Multiply with the exception of Rounding.

ROUNDING

First, let's look at the logic on the Control logic at coordinate G1/G2, quadrant B, the KV module at 7H14. The top input is m0, which would be set if doing a Round or 41 instruction. This bit would be sent to the GE modules on clock two.

Go to coordinate D9/D10, quadrant A and the GE module at 7E01. This diagram does not show it, but the KV module sends two bits to the GE module to its N-1 inputs. This module is for result bits 46 through 49 so this would be the equivalent of bit 45. Since both bits are ones, this causes a one to be carried into bit 46 of the product. This then is added, adding the Round bit as shown in the learning activity 10-A.

You have completed learning activity 10-C. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 10-D. EXERCISE: MULTIPLY UNIT TROUBLESHOOTING

The diagnostics for the CYBER 170 Model 750/760 gives you a list of modules that could cause the failure. You then change/swap them one by one until the failure is fixed.

9036H

MODULE 11
FUNCTIONAL UNIT DIAGNOSTICS

This module familiarizes you with the theory and use of the CYBER 170 Model 750/760 Functional Unit diagnostics.

PRETEST

To begin this module, sign on the PLATO terminal and take the module 11 test. Do not waste time on answers you do not know. By taking this test, you may be able to skip some of the learning activities in this module.

LEARNING ACTIVITIES

In the Assigned column below, put a check mark by each activity assigned to you by PLM and proceed through those assigned activities. Check off each activity as you complete it. You may choose to do all of these activities or do some activities more than once.

Assigned	Completed	Activity	Description	Page
_____	_____	11-A	Text Reading: Introduction to Diagnostics. This activity explains the role of the diagnostics in testing the functional units.	11-3
_____	_____	11-B	Reference Reading: CT7 Description. This activity describes the operation and use of CT7.	11-5
_____	_____	11-C	Exercise: CT7 Usage. This activity reinforces your knowledge of CT7.	11-11
_____	_____	11-D	Reference Reading: CU4 Description. This activity describes the operation and use of CU4.	11-15
_____	_____	11-E	Exercise: CU4 Usage: This activity reinforces your knowledge of CU4.	11-20

Assigned	Completed	Activity	Description	Page
_____	_____	11-F	Reference Reading: CT1 Description. This activity describes the operation and use of CT1.	11-23
_____	_____	11-G	Exercise: CT1 Usage. This activity reinforces your knowledge of CT1.	11-32
_____	_____	11-H	Reference Reading: CT1 Description. This activity describes the operation and use of DAT.	11-35
_____	_____	11-I	Exercise: DAT Usage. This activity reinforces your knowledge of DAT.	11-42
_____	_____	11-J	Reference Reading: MA5 Description. This activity describes the operation and use of MA5.	11-45
_____	_____	11-K	Exercise: MA5 Usage. This activity reinforces your knowledge of MA5.	11-50

LEARNING ACTIVITY 11-A. TEXT READING: INTRODUCTION TO DIAGNOSTICS

This activity describes the role that the functional unit diagnostics play in functional unit maintenance. It introduces you to each of the diagnostics you will be studying.

OBJECTIVE

- You will be able to identify a need for the CYBER 170 Model 750/760 Functional Unit diagnostics.

DIAGNOSTICS

Before any problem with a CYBER 170 Model 750/760 Functional Unit can be repaired, it is necessary to determine exactly where the problem is located. The diagnostics, which are a part of the Maintenance Software Library (MSL), assist you in performing this task. Effective use of these diagnostics enables you to isolate the problem, often down to a few possible modules. These diagnostics play an important role in the CYBER 170 Model 750/760 Functional Unit maintenance scheme. Although many diagnostics exist for testing of the functional units, we will discuss only CT7, CU4, CT1, DAT, and MA5.

Although these five tests are primarily used to test the functional units, other parts of the Central Processor Unit (CPU) must be used to do the testing. The instruction stack, operating register, and transmission paths must be operational to get operands to and from the functional unit being tested.

In reality, the CT7 diagnostic is not a functional unit test. It is a general test of the CYBER 170 Model 750/760 CPU. A set of operands is executed in the CPU and the results checked against a second set of answers. This second set is generated by simulation. No specific functional unit is tested. A failing unit would be indicated by the failure of an instruction which uses that functional unit.

The diagnostic, CU4, is a composite of several short tests. These tests were each written to check a specific functional unit. They were combined for ease of use. Later, you will learn that each of these short tests may still be used individually.

The diagnostic, CT1, is a test of the CYBER 170 CPU. It is in two parts. The first part executes from a PP and has the primary task of assuring that the CPU is capable of testing itself. It does this by testing the simple instructions which the CPU will use in testing itself in the second part of the test.

DAT and MA5 are diagnostics written to test the Floating Point Divide and Floating Point Multiply Functional Units respectively. When Divide fails, DAT provides information on the entire Divide sequence. This information is essential if you are to correct a Floating Point Divide problem. MA5 is an advanced test that will provide a list of the possible failing modules on certain hardware failures.

The sequence in which you execute these diagnostics depends upon the particular problem that confronts you. In this module you learn about each of these four tests and become familiar with their specific purpose. This enables you to decide on the best sequence to use for each functional unit problem. This procedure will change with each problem you encounter.

NOTE

It is assumed that you have loaded and run each of the four aforementioned diagnostics at least once prior to this time as part of previous training. If you have not, contact your course administrator to discuss course prerequisites.

You have completed learning activity 11-A. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-B. REFERENCE READING: CT7 DESCRIPTION

MICROFICHE: CT7 DIAGNOSTIC PROGRAM LISTING

A portion of this reference reading is on microfiche; therefore, have a viewer ready for use.

This activity familiarizes you with the theory and use of the CT7 diagnostic program. You learn the basic testing concepts it uses and become familiar with its keyboard commands and associated running and error displays.

OBJECTIVES

- You will be able to run the CT7 diagnostic program under various software and hardware conditions.
- You will be able to use its associated keyboard commands and interpret both running and error displays.

CT7 INTRODUCTION

Although CT7 may be used to test functional units of the CPU, its primary purpose is to check the control portion of it. Because of the large number of possible combinations of register instructions, functional units, and operands, this hardware is best tested using random numbers. These random numbers are used in the formation of instruction sequences (loops) and operands to be used by these instructions. If we have a known seed (starting number), the random number generator will produce random numbers that will always be in the same sequence. This is useful, as a particular failing loop can always be repeated.

Once the random instructions and operands are generated, the resulting set of instructions are executed twice, once by the CPU and once by an instruction simulator. These two results are compared. Depending upon the setting of some parameters, these results (if a failure is detected) are displayed, stored, or executed, and the test either halts or continues.

CT7 has many parameters available for controlling its execution. These are explained in the documentation for the test. Many of these will be explained later in the usage section of this course.

CT7 DOCUMENTATION

At this time, locate the microfiche titled CT7, and insert it in the viewer. Position the viewer at grid location B1. This should correspond to page 1 of the listing. Find the beginning of the documentation section of the listing (usually within the first four pages). Review the documentation, paying particular attention to basic philosophy, parameters, commands, and display information. Gain an understanding of the general concepts before looking at detailed flowcharts and section descriptions. Try to determine general flow of the test including testing sequence, PPVs, and channels used, Central Memory usage, and PPU/CPU residence.

When you are through studying the CT7 documentation on the microfiche, return to this activity and continue reading at the next topic, CT7 Displays.

CT7 DISPLAYS

We will now look at some actual CT7 displays. These displays were made from photographs taken of a CYBER 170 display console. These displays will familiarize you with the actual running/error displays for this diagnostic. Your success or failure in using CT7 effectively will depend on whether or not you can correctly interpret the information presented on these displays. Refer to figures 11-1 and 11-2.

```
P0= 002147
 *
                PP01    04  05  06  07  10  11
     CR OFF       *     D*   *   *   *   *

ALL REQUESTS

CP

PP
01
04      CT7= 002145
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP

R03L124-
A
```

```
      EM (NONE)
      CM (  65K)
CH00      01    02    03    04    05    06    07  10  11    12      13  A=0000
  F       -     E00   F    -00    -     -     -   E   -    -11      -

CT7= 000088


MODE = RUN                    AVAILABLE FL = 144777
BKP =                      EXCHANGE RATE = 0020

0 CT7          P= 000088      EX=   010040
             RA=  010000      FL=   023000
PC=00010321                                        EC=0000
```

Figure 11-1. CT7 Running Display

```
P0= 000037
 *
                  PP01    04   05   06   07   10   11
      CR OFF         *    D*    *    *    *    *
AC*010700

010700    46000    46000    46000    46000
010701    46000    46000    46000    46000
010702    42663    40524    45406    45340
010703    46000    46000    46000    46000
010704    61117    77776    05010    00700
010705    04000    01200    61000    46000
010706    04000    01200    61000    46000
010707    00000    00000    00000    00000

010710    00000    00000    00000    00000
010711    00000    00000    00000    00000
010712    00000    00000    00000    00000
010713    00000    00000    00000    00000
010714    00000    00000    00000    00000
010715    00000    00000    00000    00000
010716    00000    00000    00000    00000
010717    00000    00000    00000    00000

010720    00000    00000    00000    00000
010721    00000    00000    00000    00000
010722    00000    00000    00000    00000
010723    00000    00000    00000    00000
010724    00000    00000    00000    00000
010725    00000    00000    00000    00000
010726    00000    00000    00000    00000
010727    00000    00000    00000    00000

010730    00000    00000    00000    00000
010731    00000    00000    00000    00000
010732    00000    00000    00000    00000
010733    00000    00000    00000    00000
010734    00000    00000    00000    00000
010735    00000    00000    00000    00000
010736    00000    00000    00000    00000
010737    00000    00000    00000    00000
R03L124-
A
```

```
      EM (NONE)
      CM (  65K)
CH00     01     02     03     04     05     06     07  10   11     12     13   A=0000
   F      -     E00     E     E00     -      -      -   E    -     -11     -

CT7= 000037

MODE = RUN                    AVAILABLE FL = 144777
BKP =                      EXCHANGE RATE = 0020

0 CT7          P= 000037      EX=    010040
               RA=  010000    FL=    023000
PC=00003724                                        EC=0206



******

CT7     (INTEGER/MULT/OPTION)

******

NORMAL MODE

OPTIONS

WORDS IN LOOP          5

PASSCOUNT           3724

ERROR MODE       STOP ON ERROR AFTER

                 ATTEMPT TO SIMPLIFY

ERROR STOP        REGISTER ERROR

***************
```

Figure 11-2. CT7 Error Display

USING CT7

The purpose of this passage is to alert you to any pitfalls of the test and give any pointers that may be helpful in using it.

1. Because CT7 uses a CPU simulator, the possibility exists that the simulator produces an incorrect result. This possibility is quite remote, but keep it in mind.

2. If CT7 is run for about the same amount of time whenever it is used, the same series of random numbers are always executed. Two methods exist to make use of some new numbers. You can use ZIP mode; this will run more numbers through in less time and allow previously generated checksums to be verified against current results. Secondly, the random number generator seed can be changed. This results in a new series of random numbers. The seed is located in RA+213. Enter the new starting seed in these locations.

3. When one functional unit is failing intermittently, you can use the parameters at RA+217 through RA+246 to define only instructions that use the failing unit. This will produce an error in less time than running normally.

4. Using these same parameters (RA+217 through RA+246), you can eliminate all instructions that use the failing unit. This may be useful in determining if other units are failing also.

5. When it is desirable to allow CT7 to execute for a long period of time without being monitored by a CE, use the parameter at RA+177 (set = nonzero). If the parameter at RA+177 is set nonzero, any errors detected will be entered in the DAY FILE and execution of CT7 will continue.

6. Normally, parameter RA+202 (stop) is left at the default value of zero. In the case of intermittent failures, it may be advantageous to set the parameter to 3. By doing this, the failing loop can be captured and executed until it fails again. Margins could now be applied.

7. When a given pass count is known to fail, use parameter RA+211 to get to the failing loop in a hurry. This is also useful if, in the course of working on a problem, the failing loop is destroyed or otherwise lost.

8. Note the memory locations of the machine answers, simulator answers, pass count, and so forth. In the case of an intermittent, by looping without stopping and observing these locations, a toggling bit or bits sometimes can be seen.

You have completed learning activity 11-B. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-C. EXERCISE: CT7 USAGE

This activity will reinforce what you have learned about the CT7 diagnostic program in learning activity 11-B. Now that you have studied the information on CT7 you should be familiar with the test's philosophy, keyboard commands, and displays. This exercise gives you a chance to determine whether you have a good understanding of the diagnostic.

OBJECTIVE

- You will be able to explain the CT7 diagnostics, list running and error displays, and determine failing instructions.

Directions: Complete the following exercise. Use the materials from learning activity 11-B as a reference. When you have completed the exercise, compare your answers with those provided at the end of this learning activity.

1. An error is detected by CT7 when the emulated answer does not compare with the machine answer.

2. What is the result of setting the following locations as follows?

   LOC. 217 (Relative) = 1
   LOC. 220 (Relative) = 40
   LOC. 221 (Relative) = 41

   list 40 and 41 only.

3. List the four options available when CT7 detects an error.

   (102)

   0 a. emulate and stop
   1 b. stop on error
   2 c. loop on error
   3 d. loop on error and stop

4. It is desirable to begin execution of CT7 at PASS count $4,721,342_8$. How is this accomplished?

Pass count in 111

5. List the 5 different error types detected and defined by CT7.

a. Register error

b. Exp. error

c. Low shift count error

d. Instruction error

e. Mask error

6. Explain ZIP Mode.

High number of random instruction exe. executed with out simulation results checked and checked against previous checkpoint

7. It should be assumed that the simulator answer is always the correct answer. True or False? Explain your answer.

False - simulator instructions could fail

8. The documentation for CT7 states that machine and simulated answers are stored in Central Memory in exchange package format. What is exchange package format?

Looks like exchange package

9. In the case of a register type error, most of the pertinent information is displayed, such as the machine and simulator answer failing register and error conditions. How do you determine the failing instruction?

Check which instruction used
Building register that was [illegible]
[illegible]

10. Random numbers are used for instructions only. Operands are fixed by the test itself. True or False? Explain your answer.

False both instruction and operands
are randomly [illegible]

ANSWERS FOR LEARNING ACTIVITY 11-C

1. Machine, simulator (or vice-versa).

2. Only 40XXX and 41XXX instructions will occur in the random loop.

3. a. Simplify operands and instructions and stop.
   b. Stop without simplification.
   c. Loop on error.
   d. Loop on error and stop when error reoccurs.

4. Enter memory location RA+211 with the number 4721342_8.

5. a. Register error
   b. Zip error
   c. Loop checksum error
   d. Increment write error
   e. Move error
   f. Control error

6. In ZIP Mode the simulator is not used. Instead, only machine answers are generated, checked, summed, and compared against previously stored checksums. In this mode, many more sets of random numbers can be checked in less time.

7. False. In the vast majority of the cases, the simulated answer is the correct one. However, because CT7 utilizes a CPU simulator, there is a remote possibility of simulator error.

8. Exchange package format means the resulting answers are stored in memory to look exactly like an exchange package when the CPU or the simulator completes execution of a random loop.

9. The random instruction loop is located at RA + 1200 in Central Memory. The instructions should be examined to locate one that used the same result register as the failing register from the error display. Careful examination of this instruction and preceding ones should lead to the failing one.

10. False. Both instructions and operands are formed from the random number generator output.

You have completed learning activity 11-C. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-D. REFERENCE READING: CU4 DESCRIPTION

MICROFICHE: CU4 DIAGNOSTIC PROGRAM LISTING

A portion of this reference reading is on microfiche; therefore, have a viewer ready for use.

In this activity you become familiar with the theory and use of the CU4 diagnostic program. You learn the basic testing concepts it uses and become familiar with its keyboard commands and associated running and error displays.

OBJECTIVES

- You will be able to run the CU4 diagnostic program under various software and hardware conditions.
- You will be able to use its associated keyboard commands and interpret both running and error displays.

CU4 INTRODUCTION

There are two approaches to testing arithmetic hardware: using opened operands and using random operands together with a simulator. The CPU test CU4 incorporates both of these methods. When opened operands are used for data, the result of the operation can be compared to a known or canned answer for accuracy. The primary drawback with this method of testing is the limited number of answers that can be stored. This disadvantage is overcome when random data is used for input. The correct answer is now simulated using either a PPU or, with the CPU, using functional units other than the one currently being tested. CU4 uses a CPU simulator.

The test CU4 is actually a collection of small individual functional unit tests. They are combined to run as one test. Certain types of CPU failures may result in CU4 not executing properly. Better results can often be obtained by running each small test individually. The documentation on CU4 contains an explanation of how to do this. If CU4 executes properly, it can be assumed that all functional units are capable of performing their basic task.

As with any CPU diagnostic, Central Memory and exchange jump must be stable before attempting to execute CU4. False errors can result, especially when memory is shaky. Some caution must be exercised when examining errors detected by CU4. Whenever a CPU simulator is used, the possibility of an incorrect answer from the simulator exists. Although this possibility is somewhat remote, it should not be overlooked when examining test results.

CU4 DOCUMENTATION

At this time, locate the microfiche titled CU4 and insert it into the viewer. Position the viewer at grid location B1, which should correspond to page 1 of the listing. Find the beginning of the documentation section of the listing (usually within the first four pages). Review the documentation paying particular attention to basic philosophy, parameters, commands, and display information. Gain an understanding of the general concepts before looking at detailed flowcharts and section descriptions. Try to determine general flow of the test including testing sequence, PPVs and channels used, Central Memory usage, and PPV/CPU residence.

When you are through studying the CU4 documentation on the microfiche, return to this activity and continue reading at the next topic, CU4 Displays.

CU4 DISPLAYS

We will now look at some actual CU4 displays. These displays were made from photographs taken of a CYBER 170 display console. These displays will familiarize you with the actual running/error displays for this diagnostic. Your success or failure in using CU4 effectively will depend on whether or not you can correctly interpret the information presented on these displays. Refer to figures 11-3 and 11-4.

```
P0= 011450
 *
                PP01    04  05  06  07  10  11
     CR OFF      *      D*   *   *   *   *

ALL REQUESTS

CP

PP
01
04      CU4= 011446
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP

R03L124-
A
```

```
        EM (NONE)
        CM (  65K)
CH00    01    02    03    04    05  06  07 10  11  12    13  A=0000
  F     -    E00    E    E00    -   -   - E   -   E11   -

CU4= 011553

MODE = RUN                    AVAILABLE FL = 107777
BKP =                      EXCHANGE RATE = 0020

0 CU4          P= 011555     EX=   010040
             RA=  010000     FL=   060000
PC=00000025
```

Figure 11-3. CU4 Running Display

```
P0= 011247
 *
                  PP01    04  05  06  07  10  11
     CR OFF          *    D*   *   *   *   *

ALL REQUESTS

CP

PP
01
04     CU4= 011247
05
06
07
10
11     TYPE PXX. TO DUMP PP XX ON LP

R03L124-
A
```

```
      EM (NONE)
      CM (  65K)
CH00      01    02    03    04    05    06    07  10  11  12    13   A=0000
  F        -    E00    E    -00    -     -     -   F   -  -11    -

CU4= 011250

MODE = RUN                    AVAILABLE FL = 107777
BKP =                      EXCHANGE RATE = 0020

0 CU4          P= 011247      EX=   010040
               RA=  010000     FL=   060000
PC=00000023
```

Figure 11-4. CU4 Error Display

USING CU4

The purpose of this passage is to alert you to any pitfalls of the test and give any pointers that may be helpful in using it.

1. As mentioned in the documentation, certain hardware failures may make it impossible to run CU4. When this occurs, run each test separately.

2. The first test executed within CU4 is called PATH. This test attempts to check data paths to and from all functional units. In many cases, this test will fail regardless of which unit has a problem. When this occurs, enter memory location RA+113 equal to zero and run it again. CU4 will now have an error stop that points to a definite functional unit.

3. In the case of some failures, other tests should be used to assist in repair. This is specifically true in the case of a Multiply (use MA5) or Divide (use DAT) failure.

4. To use CU4 effectively, the program listing must be examined upon an error. By looking at the code immediately preceding the error stop, it can be ascertained what the contents of registers should or should not be.

5. When using the individual form of one of the tests continued on CU4, use the documentation for the individual tests as well. If you do not, memory addresses will be incorrect.

You have completed learning activity 11-D. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-E. EXERCISE: CU4 USAGE

This activity will reinforce what you have learned about the CU4 diagnostic program in learning activity 11-D. Now that you have studied the information on CU4, you should be familiar with the test's philosophy, keyboard commands, and displays. This exercise gives you a chance to determine whether you have a good understanding of the diagnostic.

OBJECTIVE

- You will be able to explain the CU4 diagnostics, list running and error displays, and determine failing instructions.

Directions: Complete the following exercise. Use the materials from learning activity 11-D as a reference. When you have completed the exercise, compare your answers with those provided at the end of this learning activity.

1. After loading CU4, Central Memory location RA+121 is set equal to zero. What effect does this have on the execution of CU4?

   incrementer unit not tested

2. CU4 halts with an error. P Register equals 13407. Which functional unit failed?

   multiply/divide

3. Documentation on each of the individual tests contained in CU4 exists twice within the MSL program listings. Why?

   so each test can be run individually

4. In its testing process, CU4 uses both fixed or canned answers and simulated answers in the testing process. True or False? Explain your answer.

True

5. Why is test PATH placed first on the CU4 diagnostic list?

test that it gets to all hardware units

ANSWERS FOR LEARNING ACTIVITY 11-E

1. The subtest INCR will not be executed as a part of CU4.

2. Multiply.

3. Because each test may be executed as part of CU4 or by itself. Addresses used will differ greatly, so two different listings are required.

4. True. An examination of CU4 documentation clearly states the use of both.

5. If there is a problem with a data path between Functional Units, it will be detected early and prevent later errors of an unknown nature.

You have completed learning activity 11-E. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-F. REFERENCE READING: CT1 DESCRIPTION

MICROFICHE: CT1 DIAGNOSTIC PROGRAM LISTING

A portion of this reference reading is on microfiche; therefore have a viewer ready for use.

In this activity you become familiar with the theory and use of the CT1 diagnostic program. You learn the basic testing concepts it uses and become familiar with its keyboard commands and associated running and error displays.

OBJECTIVES

- You will be able to run the CT1 diagnostic program under various software and hardware conditions.
- You will be able to use its associated keyboard commands and interpret both running and error displays.

INTRODUCTION TO CT1 TEST

CT1 is a fixed operand command test of the CPU. The first section of the test, called "Quick Look" checks the ability of the CPU to execute the basic instructions necessary to control a comprehensive command test. Section two "CT1CM" executes in the CPU and verifies that the CUP can execute all instructions properly. The following description explains the overall structure of CT1.

CT1 consists of the following routines.

1. CT1 - PPU program, resides in PPU 0, Controls CT1.
2. CQL - PPU program, resides in PPU1, Quick Look Test.
3. 1CQ - PPU program, overlay to CQL, Read from central memory location 60000.
4. CTC - CPU program, resides in CM from 00000 - 101100, CT1CM (Part 2) of CT1.

General Test Method - Quick Look

This test section assumes that central memory is functioning correctly. The central Processor (CP) executes one instruction and then encounters a stop code (00). The normal test sequence is: the test loads up the proper operand register with the next operand as indicated by the displayed register type and number, exchanges the CP, re-exchanges the CP obtaining the instruction result from the

Output Exchange Package, increments the CP address in the Input Exchange Package if required, checks the instruction result and resumes testing according to the selected test parameters.

General Test Method - CTVCTC

CTBCTC is subdivided into instruction sequences for test purposes. The order in which each sequence is tested is fixed, since instructions from previously tested sequences are used in checking other sequences. The order for sequence testing is as follows:

1. Boolean
2. Increment
3. Long Add
4. Branch
5. Shift
6. Floating Sum
7. Floating Multiply
8. Floating Divide

Register X1 and X2 are normally used as the initial operand registers, except for some of the instructions which require other registers, as in the Increment Sequence. Registers X3 and X4 are normally used as the test result destination register and correct or expected result destination register.

Return to the Quick Look Section is not possible once the CTVCTC section has been executed.

CT1 Sequence

The following is a list of the instructions in the order in which they occur within CT1. By referencing the attached Chart is should be possible to isolate the area of the machine not previously used, and then through the block diagrams, the possible failing pak(s).

Section I - Quick Look

1.	Exchange Jump with all 0's	See EJT for possible failing hardware components
2.	Exchange Jump with all 1's	
3.	Boolean, 10 instruction	
4.	Boolean, 14 instruction	
5.	Increment, 72 instruction	- Check X path thru F adder
6.	Increment, 62 instruction	- Check B as a result reg.
7.	Increment, 61 instruction	- Check B as a source reg.
8.	Increment, 51 instruction	- Check A as a result reg.
9.	Increment, 50 instruction	- Check A as a source reg.
10.	Increment, 52 instruction	- Check central read capability

11.	Increment, 71 instruction	- Check X central read result reg.
12.	Boolean 11 instruction	- Checks logical Prod. (X5 - X4)
13.	Boolean 13 instruction	- Checks logical Diff. (X4 - X3)
14.	Branch, 02 instruction	- Checks basic 01 jump
15.	Branch, 030 instruction	- Checks jump on X = 0
16.	Branch, 06 instruction	- Jump on B reg. compares
17.	Branch, 04 instruction	- Jump on B reg. compares
18.	Branch, 01 instruction	- Check return jump

This ends the quick look section of the test. The previously, successfully executed instructions, insure that the CM portion of the test can successfully execute in the CPU. In the CM section the instructions are fully checked out as opposed to the partial checks made in the quick-look section.

19. Boolean, 10 inst. -
20. Boolean, 14 inst. -
21. Boolean, 12 inst. -
22. Boolean, 16 inst. -
23. Boolean, 11 inst. -
24. Boolean, 15 inst. -
25. Boolean, 13 inst. -
26. Boolean, 17 inst. -
27. Increment, 70 inst. -
28. Increment, 71 inst. -
29. Increment, 60 inst. -
30. Increment, 50 inst. -
31. Increment, 61 inst. -
32. Increment, 51 inst. -
33. Increment, 62 inst. -
34. Increment, 72 inst. -
35. Increment, 52 inst. -
36. Increment, 63 inst. -
37. Increment, 73 inst. -
38. Increment, 53 inst. -
39. Increment, 64 inst. -
40. Increment, 74 inst. -
41. Increment, 54 inst. -
42. Increment, 66 inst. -
43. Increment, 76 inst. -
44. Increment, 56 inst. -
45. Increment, 65 inst. -
46. Increment, 75 inst. -
47. Increment, 55 inst. -
48. Increment, 67 inst. -
49. Increment, 77 inst. -
50. Increment, 57 inst. -
51. Long Add, 36 inst. -
52. Long Add, 37 inst. -
53. Branch, 03X inst. -
54. Branch, 04 inst. -
55. Branch, 05 inst. -
56. Branch, 06 inst. -
57. Branch, 07 inst. -
58. Shift, 20 inst. -
59. Shift, 21 inst. -
60. Shift, 43 inst. -
61. Shift, 22 inst. -
62. Shift, 23 inst. -
63. Shift, 27 inst. -
64. Shift, 26 inst. -
65. Shift, 24 inst. -
66. Shift, 25 inst. -

These increment instructions all use the sum of 2 registers.

These incr. inst. all involve the difference of 2 registers.

X=0-7

67. Floating Add, 30 inst. -
68. Floating Add, 32 inst. -
69. Floating Add, 31 inst. -
70. Floating Add, 33 inst. -
71. Floating Add, 34 inst. -
72. Floating Add, 35 inst. -
73. Floating Mult., 42 inst. -
74. Floating Mult., 40 inst. -
75. Floating Mult., 41 inst. -
76. Floating Divide, 44 inst. -
77. Floating Divide, 45 inst. -

CT1 DOCUMENTATION

At this time locate the microfiche titled CT1 and insert it in the viewer. Position the viewer at grid location B1, which should correspond to page 1 of the listing. Find the beginning of the documentation section of the listing (usually within the first 4 pages). Review the documentation paying particular attention to basic philosophy, parameters, commands, and display information. Gain an understanding of the general concepts before looking at detailed flowcharts and section descriptions. Try to determine general flow of the test including test sequence, PPUs and Channels used, Central Memory usage, and PPU/CPU residence.

NOTE

When you are through studying the CT1 documentation on the microfiche, return to this activity and continue reading at the next topic, CT1 Displays.

CT1 DISPLAYS

We will now look at some actual CT1 displays. These displays were made from photographs taken of CYBER 170 display console. The purpose of these displays is to familiarize you with the actual running/error displays for the diagnostic. Your success or failure in using CT1 effectively will depend largely on whether or not you can correctly interpret the information presented on these displays. Refer to figures 11-5 and 11-6.

CT1-I

CT1-I

52 TEST OP1 X4 = 0000 0000 0000 0000 2008
OP2 K = 0000 0000 0000 0000 0000

Figure 11-5. CT1 - Running Display

CT1-I

```
PARAMETERS. . . . SE. . . .
    CT1QL CONTROL - CPUO
    PO= 014215.

    INPUT EXCHANGE PACKAGE - 1322

P    00014214   A0   666666   B0   666666
RA   00014003   A1   666666   B1   666666
FL   00177777   A2   666666   B2   666666
EM   00006666   A3   666666   B3   666666
RE   66666666   A4   666666   B4   666666
FE   66666666   A5   007700   B5   666666
MA   66666666   A6   666666   B6   666666
*    66666666   A7   666666   B7   666666

X0   0000  0000  0000  0000  0006
X1   0000  0000  0000  0000  0006
X2   0000  0000  0000  0000  0006
X3   0000  0000  0000  0000  0006
X4   0000  0000  0000  0000  0006
X5   0000  0000  0000  0000  0006
X6   0000  0000  0000  0000  4001
X7   0000  0000  0000  0000  0015

    OUTPUT EXCHANGE PACKAGE - 1342

P    00014215   A0   666666   B0   000000
RA   00014003   A1   666666   B1   666666
FL   00177777   A2   666666   B2   666666
EM   00000000   A3   666666   B3   666666
RE   06666600   A4   004001   B4   666666
FE   66666600   A5   007700   B5   666666
MA   00666666   A6   666666   B6   666666
*    00000000   A7   666666   B7   666666

X0   0000  0000  0000  0000  0006
X1   0000  0000  0000  0000  0006
X2   0000  0000  0000  0000  0006
X3   0000  0000  0000  0000  0006
X4   5212  5252  5252  1252  5252
X5   0040  0000  0000  4000  0006
X6   0000  0000  0000  0000  4001
X7   0000  0000  0000  0000  0015
```

CT1-I

```
    ERROR STOP
52   TEST      OP1    X6 = 0000  0000  0000  0000  4001
               OP2     K = 0000  0000  0000  0000  0000
               CR        = 5252  5252  5252  5252  5252
               TR      M-X4 5212  5252  5252  1252  5252

000000  00000  00000  00000  00000
000001  00000  00000  00000  00000
000002  00000  00000  00000  00000
000003  00000  00000  00000  00000
000004  00000  00000  00000  00000
000005  02000  01434  61000  46000
000006  00141  12511  41114  41145
000007  11511  17112  22128  11301

000010  13221  34213  62140  21412
000011  14151  42114  25143  11436
000012  11021  11566  27665  16763
000013  00510  12310  40141  30000
000014  10000  46000  61000  46000
000015  00000  00000  00000  00000
000016  10011  46000  61000  46000
000017  00000  00000  00000  00000

30XX    0X     1X     2X     3X     4X     5X     6X     7X
X0      6210   0370   2007   3064   7547   7434   1401   2000
X1      0100   0100   7777   5400   5400   5400   3456   6440
X2      2745   2726   6220   3014   7477   7446   0100   5400
X3      5000   0100   2000   2000   2000   1401   3106   6314
X4      7502   4456   3023   6315   5555   3401   2000   5000
X5      1602   5000   5400   5400   5400   5000   3074   7543
X6      5400   7543   7476   6314   7427   7537   5400   6020
X7      7502   6020   2000   5000   5400   6020   7476   2000

00XX    0X     1X     2X     3X     4X     5X     6X     7X
X0      0277   0000   0001   0014   2425   0000   0000   0000
X1      0000   0000   4214   0006   2425   0000   0000   0000
X2      0000   0000   6666   0005   0062   0000   6100   0000
X3      7126   0000   6666   0004   0000   0017   7000   0000
X4      0004   0015   6666   0017   0014   1546   0000   0000
X5      4003   0000   5400   0004   0006   3041   0000   0000
X6      2534   4215   5746   4003   0004   0000   0000   0040
X7      7050   7601   0005   0002   0004   0004   0000   0000

CHANNEL STATUS - D D E E   E E E E   E E E E
```

Figure 11-6. CT1 - Error Display

CYBER 170 Model 750/760 Functional Units
Learning Activity 11-F

USING CT1

The purpose of this write-up is to alert you to any pitfalls of CT1 and give any pointers that may be helpful in using it.

1. CT1 is a stand-alone test and therefore must be loaded from the MSL initial stand-alone display.

2. CT1 does not check for memory parity errors of SECDED errors. The test assumes that Central Memory is fully operational. Exchange jump must also be operational.

3. The test uses "canned" operands, therefore, high-speed checking of random instructions are not checked. Use a test like CT7 for testing random instruction sequences.

4. If an error is indicated in a register, run section 5 along. This section uses all ones and all zeros and any error can be more easily diagnosed as to whether or not the failure is occuring in multiple registers. Section 5 can be selected by entering PP location 77 equal to 20, prior to running the CTVCTC section of the test. This is accomplished by entering 0077X0020 carriage return.

5. Use the "T" key to eliminate the CT1 data display during normal running. This will result in the test executing more rapidly. The "T" key can be used in case of an error to view the failing data, (Normal Display).

6. Note the following special error codes.

   a. Branch error codes

      CR - 1 (should branch)
           0 (should not branch)
      TR - 1 (did branch)
           0 (did not branch)

   b. Shift sequence error codes - for instructions 24, 25, 26
      Requiring a B register for part of a result;
      OP2 = 00000 000jk C0000 000jk
      jk = Test result B register count
      jk - Correct result B register count

      CR0400 RRRRRR 0000 AAAAAA
      TR0400 RRRRRR 0000 AAAAAA
      RRRRRR - Return address stored in central memory
      AAAAAA - Branch address

c. In certain cases, the same test section may be used for two central processor instructions rather than one, due to the similarity between the two instruction and between the types of operands that will most effectively test the instructions. An example is in the single (40) and double (42) precision multiply instructions test. In these particular test sections the error display would be:

42-42 (42) T4 S1A

The 42 in parentheses indicates that the instruction is the one that actually failed. The 42-40 indicates that these two instructions are being checked together in this test.

7. In case of an error, consult the test sequence list (under description) and note the instruction types executed prior to the instruction which failed. This may assist you in determining what hardware component(s) is failing.

LEARNING ACTIVITY 11-G. EXERCISE: CT1

This activity will reinforce what you have learned about the CT1 Diagnostic Program in learning activity 11-F. Now that you have studied the information on CT1 you should be familiar with the test's philosophy, keyboard commands, and displays. This exercise gives you a chance to determine whether you have a good understanding of the diagnostic.

OBJECTIVE

- You will be able to explain the CT1 diagnostic, list running and error displays, and interpret error messages.

Directions: Complete the following exercise. Use materials from learning activity 11-F as a reference. When you have completed the exercise, compare your answers with those provided at the end of this learning activity.

1. The CT1 program uses its stand-alone error display when running under control of EXC. True or False? False

2. Which section of the CPU portion of CT1 will indicate whether a failure is occuring in just one register or in multiple registers, and why?

   Section 5

3. The CTVCTC (CPU driven) portion of CT1 executes with fixed operands and instruction. True or False? True

4. The Quick Look (PP driven) portion of CT1 can be bypassed and testing will begin with the CPU driven test. What keyboard command will do this?

   BC

The answers for this learning activity are on the following page.

ANSWERS FOR LEARNING ACTIVITY 11-G

1. False

2. Section 5 because patterns of all ones and all zeros are used.

3. True

4. Enter a BC (CR) immediately after loading is completed.

LEARNING ACTIVITY 11-H. REFERENCE READING: DATC DESCRIPTION

MICROFICHE: DAT DIAGNOSTIC PROGRAM LISTING

A portion of this reference reading is on microfiche; therefore, have a viewer ready for use.

In this activity you become familiar with the theory and use of the DAT diagnostic program. You learn the basic testing concepts it uses and become familiar with its keyboard commands and associated running and error displays.

OBJECTIVES

- You will be able to run the DAT diagnostic program under various software and hardware conditions.
- You will be able to use its associated keyboard commands and interpret both running and error displays.

DAT INTRODUCTION

DAT will assist you in the repair of the Divide Functional Unit. Two types of operands are used, fixed or canned and random. Regardless of the type of operand, a simulator is employed to determine if functional unit results are correct or not.

DAT makes no effort to determine the failing component or components. It does provide the CE with information concerning the failure. This information is normally not available unless long and tedious hand calculations are performed on paper. Actual error information as displayed is well explained in the documentation section of the DAT program listing. A proper understanding of this information greatly assists in the process of scoping out a Divide Functional Unit problem.

DAT DOCUMENTATION

Locate the microfiche titled DAT and insert it into the viewer. Position the viewer at grid location B1, which should correspond to page 1 of the listing. Find the beginning of the documentation section of the listing (usually within the first four pages). Review the documentation paying particular attention to basic philosophy, parameters, commands, and display information. Gain an understanding of the general concepts before looking at detailed flowcharts and

section descriptions. Try to determine general flow of the test including testing sequence, PPVs and channels used, Central Memory usage, and PPV/CPU residence.

When you are through studying the DAT documentation on the microfiche, return to this activity and continue reading at the next topic, DAT Displays.

DAT DISPLAYS

We will now look at some actual DAT displays. These displays were made from photographs taken at a CYBER 170 display console. These displays will familizarize you with the actual running/error display for this diagnostic. Your success or failure in using DAT effectively will depend on whether or not you can correctly interpret the information presented on these displays.

```
P0= 000637                                    EM (NONE)
 *                                            CM (  65K)
              PP01    04  05  06  07  10  11  CH00    01    02   03   04   05   06   07 10  11  12    13  A=0000
    CR OFF     *     D*   *   *   *   *        F      -    E00   F   -00   -    -    -  E   -   E11   -

ALL REQUESTS                                  DAT= 000637

                                              MODE = RUN                  AVAILABLE FL = 164777
                                              BKP =                   EXCHANGE RATE = 0020

                                              0 DAT          P= 000637    EX=   010040
                                                          RA=  010000     FL=   003000
                                              PC=0000002                                  EC=0000

CP
PP
01

04      DAT= 000641
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP

R03L124-
A
```

Figure 11-7. DAT - Running Display

```
P0= 000637
 *
                PP01    04  05  06  07  10  11
     CR OFF      *      D*   *   *   *   *

ALL REQUESTS




CP

PP
01

04      DAT= 000641
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP


R03L124-
A
```

```
      EM (NONE)
      CM (  65K)
CH00     01     02    03    04    05    06    07 10  11    12     13  A=0000
  F      -     E00    F    -00    -     -     -  E   -     -11    -

DAT= 001211

MODE = RUN                    AVAILABLE FL = 164777
BKP =                      EXCHANGE RATE = 0020

0 DAT          P= 001212      EX=    010040
             RA=  010000      FL=    003000
PC=00000002                                        EC=0000

ERROR—TYPE IN CN50 (CR) AND BC (CR)
```

Figure 11-8. DAT First Error Stop Display

```
P0= 000637                                   EM (NONE)
 *                                           CM (  65K)
              PP01   04  05  06  07  10  11  CH00   01   02   03   04   05   06   07 10  11  12   13  A=0000
    CR OFF     *    D*   *   *   *   *       F     -    E00   F   E00   -    -    -  E   -  -11   -

ALL REQUESTS                                 ROUND DIVIDE FAILURE, S = 5
                                                   INSTRUCTION = 45612
                                                 XJ = 3174   6565   7503   0307   5470
                                                 XK = 5126   3154   1773   6407   3777
                                                EXI = 5533   2311   4126   4153   1122
                                                AXI = 5533   2301   4126   4153   1122
                                              FBITS = 0000   0010   0000   0000   0000
                                             BIT = 40
                                             ERRCT =  0000   0000   0000   0000
                                             BIT SHFT
                                             ITERATION = 03
                                               PICK = 1
                                               1XK = 0000   4623   6004   1370   4000
                                              REMAIN = 4437   1516   4610   3025
CP                                           XK MULTIPLE TABLE
                                                   1XK = 0000   4623   6004   1370   4000
PP                                                 2XK = 0001   1447   4010   2761   0000
01                                                 3XK = 0001   6273   2014   4351   4000
                                                   4XK = 0002   3117   0020   5742   0000
04   DAT= 000641                                   5XK = 0002   7742   6024   7332   4000
05                                                 6XK = 0003   4566   4031   0723   0000
06                                                 7XK = 0004   1412   2035   2313   4000
07                                           ITER.,PICK,REMAINDER TABLE
10                                           01   1   1742   1476   6717   1470   02   3   1126   2752   2620   0702
11   TYPE PXX. TO DUMP PP XX ON LP           03   1   4437   1516   4610   3025   04   7   2757   3127   3567   4252
                                             05   4   4454   1252   7732   2525   06   7   3127   0472   5007   1252
                                             07   5   1325   6700   0536   6525   08   2   1607   2770   2405   5252
                                             09   2   4423   3672   1074   2525   10   7   2621   4663   6427   1252
                                             11   4   3075   6615   6327   2525   12   5   1014   0131   3740   1252
                                             13   1   3314   3307   6010   6525   14   5   3200   5051   0554   1252
                                             15   5   2042   2463   6206   6525   16   3   2127   2621   5515   1252
                                             17   3   2777   4201   0577   6525

R03L124-
A
```

Figure 11-9. DAT Second Error Stop Display (Result of BC [CR])

```
P0= 001211
D*
                PP01    04  05  06  07  10  11
     CR OFF      *      D*   *   *   *   *

ALL REQUESTS
```

```
     EM (NONE)
     CM (  65K)
CH00     01    02    03    04    05  06  07 10  11  12    13  A=0000
  F      -    E00    F    -00    -   -   -  E   -  -11    -

DAT= 001211

MODE = RUN                    AVAILABLE FL = 164777
BKP =                      EXCHANGE RATE = 0020

0 DAT          P= 001212     EX=   010040
             RA=  010000     FL=   003000
PC=00000002                                   EC=0000

ERROR—TYPE IN CN50 (CR) AND BC (CR)

CP     ROUND DIVIDE FAILURE, S = 5

PP
01

04     DAT= 001211
05
06
07
10
11     TYPE PXX. TO DUMP PP XX ON LP


R03L124-
A
```

Figure 11-10. DAT Second Error Stop Display
(Result of CN50 [CR])

USING DAT

The purpose of this passage is to alert you to any pitfalls of the test and give any pointers that may be helpful in using it.

1. When DAT fails, dump the error information to the printer, if possible. A small hand program can then be entered, using BCPC, with the failing operands and instruction. Now you will have easy reference when scoping the failure.

2. If line 12 on the error display (upper display) indicates a right shift of one (to normalize) occurred, the picks as displayed will not directly match the final quotient.

3. In a situation where the first iteration produced a bad pick, one or more of the Xk multiples may be incorrect.

4. If an intermittent is indicated, loop on a small program as in preceding number one above. Store the results in memory and display them. The failing bit may toggle enough to be seen.

You have completed learning activity 11-H. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-I. EXERCISE: DATC USAGE

This activity will reinforce what you have learned about the DAT diagnostic program in learning activity 11-H. Now that you have studied the information on DAT you should be familiar with the test's philosophy, keyboard commands, and displays. This exercise gives you a chance to determine whether you have a good understanding of the diagnostic.

OBJECTIVE

- You will be able to explain the DAT diagnostic, list running and error displays, and interpret error messages.

Directions: Complete the following exercise. Use the materials from learning activity 11-H as a reference. When you have completed the exercise, compare your answers with those provided at the end of this learning activity.

1. When DAT fails, the INTERMITTENT FAILURE message is displayed. What does it mean?

Problem went away

2. The quotient is basically the picks beginning with the first pick being the upper digit, the second pick being the next digit, and so forth. If the picks are read from the iteration, pick, and remainder table in the error display, they will always equal the final quotient. True or False? Explain your answer.

False with Dec [illegible]

[illegible]

3. Part of an error message generated by DAT reads: iteration = 4. What does this mean?

iteration 4 Div a incorrect remainder

4. DAT can detect divide unit failures caused by incorrect Xk multiples. True or False? False

ANSWERS FOR LEARNING ACTIVITY 11-I

1. Originally two input operands produced an incorrect answer. However, when DAT attempted to determine the failing iteration, the failure disappeared and could not be duplicated.

2. False. If Coefficient Overflow occurred, the coefficient will be right shifted by one bit.

3. The remainder generated on the fourth iteration is incorrect. It should be compared (by scoping) to the correct one to determine the reason for the difference.

4. False.

You have completed learning activity 11-I. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-J. REFERENCE READING: MA5 DESCRIPTION

MICROFICHE: MA5 DIAGNOSTIC PROGRAM LISTING

A portion of this reference reading is on microfiche; therefore, have a viewer ready for use.

In this activity you become familiar with the theory and use of the MA5 diagnostic program. You learn the basic testing concets it uses and become familiar with its keyboard commands and associated running error displays.

OBJECTIVES

- You will be able to run the MA5 diagnostic program under various software and hardware conditions.
- You will be able to use its associated keyboard commands and interpret both running and error displays.

MA5 INTRODUCTION

This diagnostic is designed to assist in repair of the CYBER 170 Model 750/760 Multiply Functional Unit. Basically, MA5 automates the normal procedure for determining the failing module. This is accomplished by converting the Multiply Unit Coefficient Troubleshooting Chart found in the logic diagrams into software lookup tables. A table lookup is performed based on error data, and the results displayed.

The failing bit is determined through the use of a simulator. Multiply Unit answers are compared with simulator answers for each set of operands to be tested. Any difference detected will result in an error display. In addition to operands, answers, and difference information, a list of all the modules that could have caused the failure are displayed. The error display includes a pointer to the module list that points to the modules more likely to have caused the failure.

NOTE

There are several precautions and notes included in the MA5 Documentation. Read these over carefully before using the test.

MA5 DISPLAYS

We will now look at some actual MA5 displays. These displays were made from photographs taken of a CYBER 170 display console. These displays will familiarize you with the actual running/error displays for this diagnostic. Your success or failure in using MA5 effectively will depend on whether or not you can correctly interpret the information presented on these displays.

```
P0= 000763                                    EM (NONE)
 *                                            CM (  65K)
                PP01   04  05  06  07  10  11  CH00   01    02   03   04   05  06  07 10  11  12   13  A=0000
    CR OFF       *    D*   *   *   *   *       F     -    E00   F   -00   -   -   -  E   -  -11   -

ALL REQUESTS                                  MA5= 000766

                                              MODE = RUN                  AVAILABLE FL = 162777
                                              BKP =                    EXCHANGE RATE = 0020

                                              0 MA5        P= 000766      EX=   010040
                                                          RA=  010000     FL=   005000
                                              PC=00000005                                EC=0000
CP
PP
01

04      MA5= 000767
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP


R03L124-
A
```

Figure 11-11. MA5 Running Display

```
P0= 000037
 *
                    PP01    04  05  06  07  10  11
      CR OFF         *     D*E   *   *   *   *

ALL REQUESTS

CP

PP
01

04      MA5= 000037
05
06
07
10
11      TYPE PXX. TO DUMP PP XX ON LP


R03L124-
A
```

```
      EM (NONE)
      CM (  65K)
CH00     01     02    03    04    05   06   07 10  11   12     13   A=0000
  F      -     E00    F    -00    -    -    - E   -   -11     -

MA5= 000037

MODE = RUN                    AVAILABLE FL = 162777
BKP =                    EXCHANGE RATE = 0020

0 MA5          P= 000037      EX=   010040
               RA=  010000     FL=   005000
PC=0000002                                    EC=0001


MULTIPLY FAILURE,SECT 03
      INSTRUCTION = 42612
       XJ = 2000     0000     0000     0167     0000
       XK = 2000     0000     0000     0177     0000
      EXI = 2000     0003     5411     0000     0000
      AXI = 2000     0003     5411     0000     0000
    FBITS = 0000     0000     2010     0000     0000
   ERRCT = 0000     0000      0000     0000     0001
TRY PASS 2 BIT = 27
NO SHIFT
PASS 2
 GV  I05   GS   H05   GK   F01
 GI  B05   GC   B06
PASS 1
 GM  G05   GE   F04   GE   E03
 GB  B09   GB   A08
OTHERS
 GJ  B03   GJ   B04   GM   G04
 GE  F03   GE   E02   GM   G06
 GE  F05   GE   E04
SECTION BITS
 GT  H03   GT   H04   GS   H05
 GR  H06   GR   H07   GR   H08
```

Figure 11-12. MA5 Error Display

USING MA5

The purpose of this passage is to alert you to any pitfalls on the test and give any pointers that may be helpful in using it.

1. The correct answer for any two Multiply operands can be determined by using the simulation entry point. This can be useful when a unique pair of operands causes a failure. (These could come from another test.)

2. Boolean, shift, long add, and increment units are used by MA5. Make sure they are operational before attempting to use MA5.

3. Before using MA5, be sure to read carefully the section titled Other Considerations in the documentation for the test.

This completes learning activity 11-J. You may review this material or continue with the next learning activity.

LEARNING ACTIVITY 11-K. EXERCISE: MATC USAGE

This activity will reinforce what you have learned about the MA5 diagnostic program in learning activity 11-J. Now that you have studied the information on MA5, you should be familiar with the test's philosophy, keyboard, commands, and displays. This exercise gives you a chance to determine whether you have a good understanding of the diagnostic.

OBJECTIVE

- You will be able to explain the MA5 diagnostics, list running and error displays, and interpret error messages.

Directions: complete the following exercise. Use the materials from learning activity 11-J as a reference. When you have completed the exercise, compare your answers with those provided at the end of this learning activity.

MA5 EXERCISE

1. MA5 has one major advantage to the CE that other tests do not offer. What is it?

   list possible failing module

2. Which functional units, other than Multiply, must be operational for MA5 to execute properly? Why?

   Program loop with increment and
   loop add write

3. How is a simulator type failure detected?

4. MA5 is capable of detecting errors in the input operand holding registers. True or False?

5. How, by using MA5, can the correct answer be determined for any two floating point numbers to be multiplied?

ANSWERS FOR LEARNING ACTIVITY 11-K

1. Once a failure is detected, a list of the possible failing modules is displayed.

2. Boolean, shift, long add, increment; they are used for simulation.

3. Two passes are made of each set of operands. For the second pass, the two are reversed (Xj and Xk). If the results of these two passes differ (they should not) a simulator failure has occured.

4. False. (So stated in the documentation.)

5. Enter the operands in Central Memory locations RA+114 and RA+115, and start execution at RA+200. the correct result can be found in RA+127. The machine answer will be found in RA+130.

MODULE 12
TROUBLESHOOTING THE FUNCTIONAL UNITS

This module permits you hands-on experience in running and interpreting diagnostics and in troubleshooting instructor-induced component failures. These failures will consist of problems in the data or control logic of any functional unit.

DIRECTIONS

This module requires an instructor and laboratory equipment for all the learning activities. Approximately 8 hours should be allowed for the laboratory activities, none of which should last more than 8 hours at a time. Contact your course administrator to schedule your lab. You will work in a group of not more than three students. Read and observe the safety practices, and also read through the procedure portion of each learning acitvity prior to doing that activity.

Each learning activity contains a problem worksheet with instructor checkpoints. Have your instructor initial the checkpoints as you complete each problem. When you have completed all the problem worksheets for that learning activity, ask your instructor to initial the learning activity instructor checkpoint on the module title page.

Because this module uses a lab instructor to verify your activities there is no test on the objectives. Thus, you do not have to return to PLATO for the remainder of the course.

This module ends with an Instructor Confirmation page. When you have completed all the learning activities, ask your lab instructor to sign it. Then remove this page from the student manual; fold and staple it as shown on the back of the page. Since the form is received by Engineering Services Education, an ESE administrator will award you mastery of this module by entering the information in your PLATO file.

SAFETY PRACTICES

While working on computer system equipment, all engineers are expected to follow reasonable and appropriate precautions with respect to electrical, mechanical, and personal safety hazards. You should pay careful attention to all entries in the maintenance documentation labeled *DANGER* or *WARNING*, which identify

hazardous areas or procedures encountered during system equipment maintenance. Follow these safety practices when working on equipment:

1. You are responsible for ensuring that no action on your part causes unsafe conditions that may expose customer personnel to hazards in any device.

2. You should never work alone on equipment having exposed operating mechanical parts or exposed hazardous power components. If you must do so, notify your EIC or manager. In any case, the following precautions must be observed:

   a. Someone familiar with the power-off controls must be in the immediate area.

   b. Personal jewelry (rings, wristwatches, bracelets, necklaces, and so forth) shall be removed. A small box in the CE tool kit makes a good storage place for these items.

   c. If using only one hand, keep the other free hand in your pocket.

   d. Avoid wearing loose articles of clothing that can be snagged and drawn into moving machinery. Roll sleeves above the elbow or wear short-sleeve shirts. Neckties, if required, should be tucked in between the second and third shirt button or fastened about 3 inches from the end with a tie tack or tie clasp, preferably nonconductive. Do not use tiechains. Clip-on neckties are preferable, as they will pull free if caught, without causing injury.

   e. Before starting equipment, ensure that no other CE or customer personnel are in a position where they might get hurt.

   f. While working on equipment, put red tape strips across any power controls, or use "do not operate" tags where available.

3. Replace worn or broken tools or test equipment as quickly as possible.

4. If the machine is running, do not reach into the works; remember, they are your fingers.

5. If using a strobe light on mechanical devices, do not touch anything; what appears stationary could be moving.

6. Safety glasses or goggles must be used if you are:

   a. Driving pins, riveting, swaging, or engaged in similar activities.

   b. Using an electric drill, grinder, reamer, and so forth.

   c. Installing or removing springs under tension or compression.

   d. Using any type of solvent, spray, or chemical for cleaning or touch-up painting.

   e. Performing any other activity that could endanger your eyes. They are your eyes, and you need them for this type of work.

7. When lifting, use a method that will not injure the spine or strain back muscles.

Above all, use good judgment and common sense - a moment of deliberate thought before you act can save you hours of agonizing afterthought.

LEARNING ACTIVITIES

You must complete this learning activity. After completing this activity, ask you lab instructor to initial the instructor checkpoint.

Instructor Checkpoint	Activity	Description	Page
_____	12-A	Lab: Troubleshooting the CYBER Model 750/760 Functional Units. During this activity, you learn how to use the diagnostics and logic diagrams to solve functional unit logic failures.	12-5

LEARNING ACTIVITY 12-A. LAB: TROUBLESHOOTING THE CYBER 170 MODEL 750/760 FUNCTIONAL UNITS

During this activity you will troubleshoot instructor-induced component failures of the CYBER 170 Model 750/760 functional units, using the logic diagrams and diagnostic error messages.

OBJECTIVE

- The student will be able to isolate failures in the CYBER 170 Model 750/760 functional units as indicated by the diagnostics.

REFERENCES

Title	Publication Number
CDC CYBER 170 Models 750/760 Functional Units Hardware Maintenance Manual	GF60420300W
MSL 100 Off-Line Maintenance System Library Reference Manual	60455770

Title	ESE Publication Number
MSL Diagnostic Program Listings CT3 DAT CU4 MA5	Part of the student manual

EQUIPMENT/TOOLS

- Standard site tools
- Minimum CYBER 170 Model 750/760 computer system configuration
- Operating system with CML installed containing all CYBER 170 Model 750/760 diagnostics
- Off-Line Maintenance System Library (CSME)
- Microfiche viewer

PROCEDURES

1. Have your instructor show you all the equipment that will be used during this learning activity.

2. During this activity you will be required to troubleshoot a minimum of 10 problems.

3. After your instructor places the problem in the mainframe, observe the equipment. Then load and, while referring to the logic diagrams, run the diagnostics in a sequence that successfully isolates the problem.

4. Fill out a problem worksheet for each problem that you troubleshoot. Follow these directions when filling out each of the sections of the problem worksheet.

   a. Failing Diagnostics: List the diagnostic(s) that are failing.

   b. Significant Error Indications: List any diagnostic error messages or conditions, which can include Status and Control Registers and any LED indicators such as a P Register.

   c. List Suspected Failing Logic: Using the diagnostics and your logic diagrams, list all the modules that could possibly cause the problem.

   d. Bad Module: Once the problem has been solved, list the pak by type and location.

   e. Comments: You may wish to make notes regarding the problem and the approach you took to solving it. It is a good idea to review the problem with your lab partner(s) and instructor.

   f. Instructor Checkpoints: Always have your instructor initial your problem worksheet after you solve the problem.

PROBLEM WORKSHEET
PROBLEM 1

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 2

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 3

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 4

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 5

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 6

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 7

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 8

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 9

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

PROBLEM WORKSHEET
PROBLEM 10

Bad Module:
Module Serial Number:
Failing Location:
Failing Diagnostics:
Significant Error Indications:

List of Possible Failing Modules:

Comments:

COMMENT SHEET

MANUAL TITLE CYBER 175 FUNCTIONAL UNITS VOLUME 2

PUBLICATION NO. 75445737 REVISION A Prod. No. R0502

FROM: NAME:

BUSINESS ADDRESS:

COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual.

CUT ALONG LINE

AA3419 REV. 11/69 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Engineering Services Education
P.O. Box O
Minneapolis, MN 55440

ATTN: EDUCATION QA
NTHESE

FOLD FOLD

CUT ALONG LINE

CORPORATE HEADQUARTERS P.O. BOX 0 MINNEAPOLIS, MINNESOTA 55440